CÁSSIA YURI TATIBANA

ESTUDO EXPERIMENTAL DO LINUX COMO PLATAFORMA PARA APLICAÇÕES DE TEMPO REAL BRANDO

FLORIANÓPOLIS
2002

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTUDO EXPERIMENTAL DO LINUX COMO PLATAFORMA PARA APLICAÇÕES DE TEMPO REAL BRANDO

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

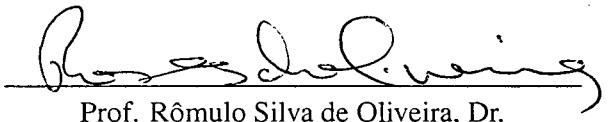
CÁSSIA YURI TATIBANA

Florianópolis, março de 2002.

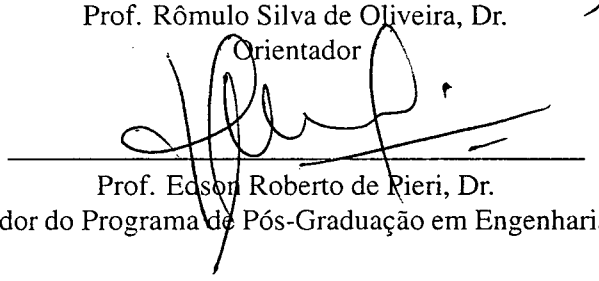
ESTUDO EXPERIMENTAL DO LINUX COMO PLATAFORMA PARA APLICAÇÕES DE TEMPO REAL BRANDO

Cássia Yuri Tatibana

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'



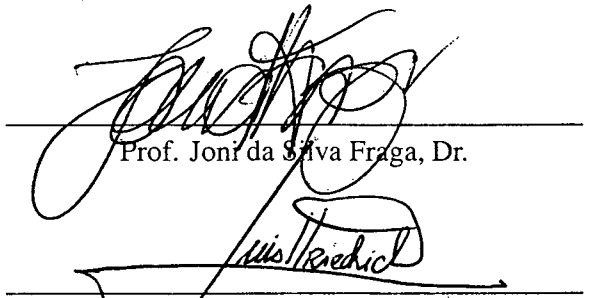
Prof. Rômulo Silva de Oliveira, Dr.
Orientador



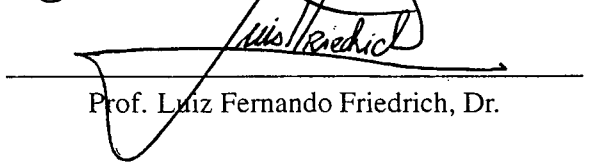
Prof. Edson Roberto de Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

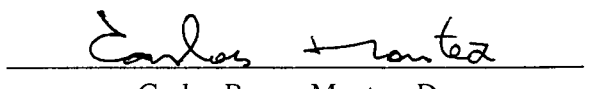
Prof. Rômulo Silva de Oliveira, Dr.
Presidente



Prof. Joni da Silva Fraga, Dr.



Prof. Luiz Fernando Friedrich, Dr.



Carlos Barros Montez, Dr.

AGRADECIMENTOS

Agradeço a meus pais pelo apoio e incentivo essenciais para conclusão deste trabalho.

Ao meu orientador Rômulo Silva de Oliveira, pela dedicação, conselhos, ensinamentos e acima de tudo paciência.

A todos amigos do LCMI, em especial Tatiana, Lu, Silvia, Alysson, e todos aqueles que tornaram tão especial o período de mestrado.

A meus amigos queridos que deixei tão longe mas que nunca estiveram distantes, Elaine, Flávio, Fernando, Rafael, Maurício, Flávio B., Edson.

A meus professores.

A CAPES pelo suporte financeiro.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

ESTUDO EXPERIMENTAL DO LINUX COMO PLATAFORMA PARA APLICAÇÕES DE TEMPO REAL BRANDO

Cássia Yuri Tatibana

Março/2002

Orientador: Rômulo Silva de Oliveira
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Sistemas de Tempo Real, Linux, Sistema Operacional
Número de Páginas: xiii + 115

A flexibilidade e o custo do Linux padrão têm atraído desenvolvedores e projetistas de sistemas de tempo real brando. É consenso que o Linux padrão não é apropriado para aplicações de tempo real crítico. Entretanto, não está claro na literatura até que ponto o Linux pode ser utilizado em aplicações com restrições temporais menos rigorosas (soft real-time). O objetivo deste trabalho é a observação do comportamento temporal de aplicações de tempo real brando sobre o Linux convencional. A observação e análise do comportamento foi feita através de tempos de resposta capturados durante a execução de aplicações hipotéticas especialmente projetadas para essas experiências. Durante o processo de construção e execução das aplicações de tempo real, alguns mecanismos capazes de melhorar o comportamento dessas tarefas foram apontados. A análise do resultado destas atividades permitiu a obtenção de informações úteis para desenvolvedores no julgamento da adequação do sistema às necessidades temporais específicas de suas aplicações.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

EXPERIMENTAL STUDY OF LINUX AS A PLATFORM FOR SOFT REAL-TIME APPLICATIONS

Cássia Yuri Tatibana

March/2002

Advisor: Rômulo Silva de Oliveira
Area of Concentration: Control, Automation and Industrial Computing
Key words: real time systems, Linux, real time tasks
Number of Pages: xiii + 115

The flexibility and low cost of standard Linux have attracted soft real-time systems developers and designers. It is common sense that standard Linux is not suitable to hard real-time applications. However, it is not strictly known in the literature the limitations of using Linux in soft real-time applications. The objective of this work is the observation of soft real-time applications temporal behavior running on standard Linux. Observation and analysis of this behavior is made through response time captured during the execution of hipothetic applications specially designed for this experiences. During the real-time applications construction and execution process, some mechanisms capable of improving the tasks behavior were indicated. The analysis of these activities results allowed the profit of useful informations to developers on judging the appropriateness of the system to the specific requirements of their applications.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Tempo real é um termo que caracteriza uma classe específica de sistemas. Nesta classe estão contidos desde jogos de videogame até sistemas de controle de tráfego aéreo. Mesmo agregando aplicações com finalidades bastante diferentes, sistemas de tempo real (STR) têm uma preocupação em comum: o tempo.

Nestes sistemas, o tempo é tratado de maneira explícita. De modo geral, é assumido que STR são sistemas capazes de oferecer garantias de correção temporal para o fornecimento de todos os seus serviços que apresentam restrições temporais [10].

Uma vez que englobam uma vasta gama de aplicações, sistemas de tempo real são divididos em crítico (*Hard Real-Time*) e brando (*Soft Real-Time*). Dentre as mais conhecidas aplicações de tempo real crítico estão aquelas de controle de plantas nucleares e sistemas de defesa militar, enquanto que aplicações multimídia e jogos de videogame constituem STR brando.

Sistemas de tempo real têm conquistado um espaço cada vez maior em todas as áreas de atuação humana. Embora a presença destes sistemas seja mais evidente em aplicações com requisitos temporais crítico, aplicações simples que implementam sistemas de tempo real brando são intensamente exploradas.

O atendimento de requisitos temporais não depende somente do código da aplicação, mas também da colaboração do sistema operacional no sentido de permitir previsibilidade ou pelo menos um desempenho satisfatório deste tipo de sistemas [10]. A previsibilidade é um conceito de extrema importância para sistemas de tempo real, mas ao mesmo tempo em que se busca previsibilidade, a flexibilidade e o custo são fatores de influência sobre a escolha da plataforma adotada para o desenvolvimento e utilização destes sistemas.

O Linux é um sistema operacional de propósito geral cuja aplicabilidade vem se mantendo em constante crescimento. Sendo software livre, com código fonte disponível e milhares de desenvolvedores em potencial espalhados por todo o mundo, esta plataforma tem sido adotada para o desenvolvimento de uma grande diversidade de sistemas. A flexibilidade oferecida contribui para sua utilização em um grande número de aplicações. Inclusive, muitos desenvolvedores empregam o Linux padrão como plataforma para aplicações de tempo real brandos. Fazem isso sem um conhecimento detalhado do comportamento temporal a ser esperado do kernel.

O atendimento a restrições de natureza temporal não está entre os objetivos de projeto do Linux. Sendo um kernel convencional, é consenso que o Linux não é apropriado para aplicações de tempo real crítico. Porém, não existe na literatura informações suficientes que permitam determinar sua aplicabilidade quando se trata de sistemas de tempo real brando. Uma vez que estes sistemas caracterizam uma grande quantidade de aplicações, os requisitos exigidos das mesmas divergem em vários aspectos.

Apesar de existirem inúmeras versões, modificações e *patches* de Linux que oferecem melhorias em determinados fatores relacionados ao comportamento temporal deste sistema, a preferência de desenvolvedores reside no uso do Linux padrão. Além da maior facilidade na instalação e manutenção do sistema, a atualização de *drivers* de dispositivos e demais utilitários realizados a cada nova versão do Linux nem sempre ocorre para suas variações.

Este trabalho objetiva disponibilizar informações a respeito do comportamento temporal do Linux na execução de tarefas de tempo real brandas. A partir de experimentações, busca-se delinear as limitações deste sistema e prover dados suficientes para que possam ajudar desenvolvedores na decisão sobre a aplicabilidade do Linux em vista dos requisitos específicos de suas aplicações. Nos experimentos realizados foi utilizado o kernel padrão 2.4.0-test10.

1.1 Estrutura da Dissertação

No capítulo 2 são descritos os conceitos relacionados a sistemas de tempo real utilizados neste trabalho. É apresentado neste capítulo; a definição e classificação de sistemas de tempo real, a descrição de tarefas periódicas e da abordagem de escalonamento adotada nos experimentos do trabalho e a problemática de tempo real.

O capítulo 3 traz uma breve revisão sobre o Linux. São abordados apenas alguns dos aspectos mais diretamente relacionados com o trabalho. Informações gerais a respeito deste

sistema operacional são ilustradas com o intuito de apontar as principais propriedades desta plataforma que acabam influenciando os experimentos realizados.

As observações relatadas durante a concepção das aplicações e cenários utilizados no trabalho são detalhadas no capítulo 4. São apresentadas também informações referentes aos meios pelos quais foram implementadas as aplicações e que acabam por afetar a maneira com que os resultados são ilustrados.

O capítulo 5 descreve os resultados obtidos pelos experimentos realizados. Através de gráficos e tabelas é verificado o comportamento temporal das tarefas implementadas, e uma análise comparativa sobre os mesmos é apresentada.

O capítulo 6 apresenta conclusões a respeito de todos os aspectos observados neste trabalho, trazendo também algumas sugestões de trabalhos futuros.

Capítulo 2

Sistemas de Tempo Real

Na realização de operações de controle de tráfego aéreo, em jogos de videogame ou na supervisão de plantas nucleares, é exigido do sistema computacional que os gerencia um comportamento imediato e acima de tudo, previsível. Para estes tipos específicos de aplicações, é importante que o sistema forneça respostas dentro de um prazo específico, sob pena de provocar danos até mesmo catastróficos ao ambiente com o qual interagem. São identificados nestes sistemas, requisitos de natureza temporal; as respostas exigidas dos mesmos não são apenas resultados esperados, são também requeridas no momento correto.

Esta é a classe de sistemas denominada de sistemas de tempo real. Em geral aplicações desta classe são também aplicações reativas, que estabelecem comunicação com o ambiente caracterizando sistemas interativos. Por isso, o conceito de sistemas de tempo real os define como sistemas que devem reagir a estímulos oriundos do seu ambiente em prazos específicos [10].

A exigência do cumprimento de requisitos temporais faz com que o aspecto de correção destes sistemas envolva correção lógica (*correctness*) e correção temporal (*timeliness*) [10].

Uma generalização do conceito de sistemas de tempo real os define como sistemas capazes de oferecer garantias de correção temporal para o fornecimento de todos os seus serviços que apresentem restrições temporais. Nesta definição, se encaixam os sistemas de tempo real que fornecem serviços a um usuário humano ou a equipamentos[10].

Independente do fato de ser reativo ou não, o sistema de tempo real possui como conceito principal, a previsibilidade. Um sistema é dito previsível no domínio lógico e temporal quando seu comportamento pode ser antecipado antes de sua execução; quando seu comportamento independe de variações do ambiente de execução provocados por características de hardware, carga ou falha.

Para que um sistema possa ser dito previsível, é preciso considerar duas hipóteses sobre o ambiente externo: a hipótese de carga e a hipótese de falhas. A primeira, determina a carga máxima gerada pelo ambiente num intervalo mínimo de tempo. Enquanto a segunda descreve tipos e freqüência de falhas às quais o sistema está exposto durante sua execução ao passo que permanece atendendo seus requisitos temporais e funcionais[12].

A partir da determinação destas hipóteses, é assumido que um sistema previsível deve ter seu comportamento antecipado mesmo quando executa sob a pior situação de carga ocorrendo simultaneamente com a hipótese de falha.

A garantia de previsibilidade exige ainda o conhecimento detalhado sobre o comportamento temporal do sistema em relação a todos os elementos que fazem parte de sua execução (linguagem de programação, hardware, suporte). Estas propriedades caracterizam a previsibilidade determinista, que permite uma análise em tempo de projeto do comportamento temporal do sistema diante dos recursos disponíveis.

Nesses sistemas, o desempenho do caso médio não é mais importante que o desempenho do pior caso. A velocidade de processamento não garante, sozinha, bom comportamento de uma aplicação de tempo real, pois este é verificado quando é possível garantir o atendimento de requisitos temporais de cada uma das atividades da aplicação[24].

Quando não é possível a antecipação completa da carga e hipótese de falhas, pode ser ainda obtida uma previsibilidade probabilista fazendo uso de simulações na obtenção de informações sobre o comportamento provável do sistema.

Quanto à garantia no atendimento de requisitos temporais, sistemas podem fornecer garantia em tempo de projeto ou garantia dinâmica. A garantia em tempo de projeto está relacionada a previsibilidade determinista, enquanto a garantia dinâmica é a garantia que pode ser obtida ou não por abordagens de melhor esforço em que a carga é dinâmica e conhecida somente em tempo de execução [19, 23].

2.1 Classificação de Sistemas de Tempo Real

Tendo em vista a aplicabilidade destes sistemas, eles se apresentam numa grande variedade quanto à complexidade, criticalidade e tamanho. Sistemas de tempo real podem ser encontrados em aplicações simples e pequenas como controladores embutidos em eletrodomésticos até aplicações complexas e robustas, como aquelas implementadas para controle de tráfego aéreo [8, 25] .

Por isso diversas classificações são encontradas para estes sistemas, cada uma delas considerando diferentes aspectos. Sob o ponto de vista da segurança, sistemas de tempo real podem ser crítico (*Hard Real Time Systems*) ou brando (*Soft Real Time Systems*). Esta classificação observa as conseqüências de uma falha temporal do sistema; quando a falha é da mesma ordem de grandeza que os benefícios do mesmo em operação normal, o sistema é caracterizado como sistema de tempo real brando. Por outro lado, se a conseqüência de uma falha temporal excede em muito os benefícios do sistema sob condições normais de funcionamento, este sistema é classificado como sistema de tempo real crítico [10].

Exemplos clássicos de sistemas de tempo real brando são aqueles voltados para aplicações de teleconferência, videogames, etc. Entre os sistemas de tempo real crítico se destacam aqueles de controle de tráfego aéreo, controle de plantas nucleares ou sistemas militares de defesa[10].

Os sistemas de tempo real críticos são ainda divididos em Sistemas de Tempo Real Críticos Seguros em Caso de Falha e Sistemas de Tempo Real Críticos Operacionais em Caso de Falha. No primeiro caso, um ou vários estados seguros podem ser alcançados na presença de falha. No segundo, entretanto, a ocorrência de falhas parciais provoca degradação do sistema, que passa a fornecer algum tipo de serviço mínimo.

A distinção de sistemas de tempo real de acordo com o ponto de vista da implementação também é definida por alguns autores como uma classificação que define sistemas de resposta garantida (em que existe recursos suficientes para suportar a carga de pico e cenários de falhas definidos) e sistemas de melhor esforço (em que a estratégia de alocação dinâmica de recursos se baseia em estudos probabilístas sobre a carga esperada e os cenários de falhas aceitáveis).

2.2 O problema de Tempo Real e Abordagens de Solução

O problema de tempo real consiste em especificar, verificar e implementar sistemas ou programas, que mesmo em situações de limitações de recursos, apresentam comportamento previsível, atendendo restrições temporais impostas pelo ambiente ou pelo usuário[10]. Considerando estes aspectos de construção, tempo real pode ser visto inicialmente como um problema intrínseco de programação concorrente.

Dentro da discussão sobre o tratamento da concorrência surgiram duas abordagens diferentes: a abordagem síncrona e a abordagem assíncrona.

A abordagem síncrona é mais abstrata em relação aos aspectos de implementação, o que é satisfatório do ponto de vista da portabilidade. Seu principio básico considera que os

cálculos e as comunicações não levam tempo. Nesta abordagem, o tempo se apresenta com granularidade suficientemente grossa para tornar sua hipótese verdadeira. A observação dos eventos é cronológica, permitindo a eventual simultaneidade entre eles. A partir das suas premissas, a concorrência é resolvida sem o entrelaçamento de tarefas e o tempo não é tratado de maneira explícita. Esta abordagem é considerada como orientada ao comportamento da aplicação e a sua verificação [10].

A abordagem assíncrona trata a ocorrência e percepção de eventos independentes numa ordem arbitrária porém não simultânea. Ela objetiva a descrição, o mais exata possível de um sistema, baseada na observação, durante a execução, de todas as combinações de ocorrência de eventos. Por se tratar de uma abordagem orientada a implementação, leva em consideração durante a especificação e projeto, características do suporte de software e hardware da aplicação. Portanto, a portabilidade é comprometida e ao mesmo tempo as considerações a respeito de características do sistema e de implementação inserem complexidade e indeterminismo [10].

A abordagem assíncrona está fundamentada no tratamento explícito da concorrência e do tempo de uma aplicação em tempo de execução; a questão do escalonamento de tempo real é o ponto principal do estudo da previsibilidade dos sistemas de tempo real [10].

2.3 Modelos de Tarefas

O conceito de tarefa é uma das abstrações básicas que fazem parte do problema de escalonamento. Tarefas ou processos são termos associados a unidades de concorrência em um sistema, unidades de processamento seqüencial que concorrem sobre recursos computacionais [8, 10].

A definição de um modelo de tarefas é determinado pela imposição de restrições temporais e relações de precedência e exclusão. Ele é parte integrante do problema de escalonamento e contribui na análise de comportamento das atividades cooperantes de uma aplicação de tempo real visando a obtenção de alguma previsibilidade [10, 19].

Tarefas são habilitadas para executar a partir da ocorrência de um evento, tal como término de uma outra tarefa, ou pela passagem do tempo [8]. De acordo com a regularidade de ativações da tarefa, ela pode ser classificada como periódica, aperiódica ou esporádica.

Tarefas periódicas são aquelas cujas ativações de processamento ocorrem indefinidamente, de forma que exista uma única ativação dentro de um intervalo regular de tempo (período). Cada ativação descreve uma instância da tarefa.

Tarefas aperiódicas ou assíncronas apresentam ativações de processamento como resposta a eventos externos ou internos, caracterizando ativações aleatórias. Um subconjunto destas tarefas, tarefas esporádicas, corresponde a um subconjunto de tarefas que possui um intervalo mínimo entre ativações consecutivas.

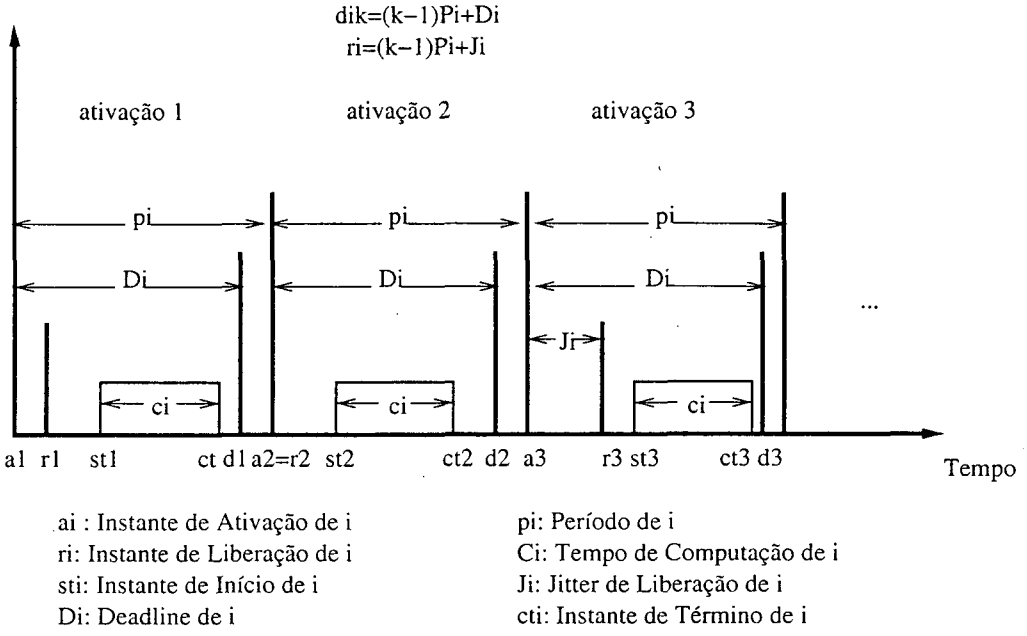
Pelo ponto de vista da previsibilidade, devido à possibilidade de análises de escalonabilidade em tempo de projeto, tarefas periódicas e esporádicas são usualmente utilizadas na implementação de tarefas críticas. Enquanto que tarefas aperiódicas são empregadas na implementação de tarefas de tempo real brando.

Além do período de tarefas de tempo real, outras restrições temporais fazem parte da descrição de seu comportamento no tempo.

- Tempo de Computação (*Computation Time*) (*C*): Tempo de uso real do processador e outros recursos do sistema para a execução completa da tarefa.
- Instante de Início (*Start Time*) (*st*): Instante em que a tarefa inicia o processamento.
- Instante de Término (*Completion Time*) (*ct*): Instante de tempo em que a tarefa é concluída. Caso a tarefa, ao ganhar o processador, executa sem interrupções, o tempo de computação (*C*) corresponde ao intervalo entre o tempo de início st e o tempo de conclusão *ct*.
- Instante de Chegada (*Arrival Time*) (*a*): Tempo em que o escalonador toma conhecimento da ativação de uma tarefa.
- Instante de Liberação (*Release Time*) (*r*): Instante em que a tarefa é incluída na fila de tarefas prontas para executar pelo escalonador do sistema.
- Jitter de Liberação*Release Jitter*(*J*): Máxima variação dos instantes de liberação das instâncias da tarefa.

Geralmente, é assumido que *a* e *r* coincidem. Ou seja, tão logo a tarefa seja acionada, ela é imediatamente liberada e inserida na fila de tarefas aptas para serem executadas (*Ready Queue*) Isso nem sempre ocorre, como é o caso de tarefas ativadas por mensagens em que o bloqueio na recepção de uma mensagem pode retardar a liberação da tarefa, ou no caso de escalonador ativado a cada período de tempo.

Conforme pode ser visto na figura 2.1, o comportamento de uma tarefa periódica T_i pode ser descrito pela quádrupla (J_i, C_i, P_i, D_i). Observa-se que o período (P) e o deadline (D) da tarefa são intervalos medidos a partir de cada início de período. Enquanto o deadline absoluto (d) e tempo de liberação (r) de cada ativação da tarefa são determinados a partir de períodos anteriores.



ai : Instante de Ativação de i
ri: Instante de Liberação de i
sti: Instante de Início de i
Di: Deadline de i

pi: Período de i
Ci: Tempo de Computação de i
Ji: Jitter de Liberação de i
cti: Instante de Término de i

Figura 2.1
Restrições temporais da tarefa periódica.

2.4 Escalonamento de Tempo Real

O escalonamento é o processo de alocação de recursos do sistema e a ordenação das tarefas na fila de pronto numa escala de execução. O escalonador é o componente do sistema responsável pela gerência dos recursos e pela implementação da solução de escalonamento empregados. Ele implementa, durante a execução, a solução de escalonamento do sistema [8, 10].

Uma solução de escalonamento mostra como o problema de alocar recursos às tarefas pode ser abordado ou mesmo resolvido. O escalonamento pode ser feito tanto em tempo de projeto quanto em tempo de execução. Soluções mistas podem ainda combinar parte do escalonamento em tempo de projeto e parte do escalonamento em tempo de execução [8].

O algoritmo de escalonamento de sistemas de tempo real deve determinar, para um conjunto de tarefas, se existe uma escala para executar as tarefas de forma a satisfazer seus requisitos de recursos e suas restrições temporais [19]. Tal verificação é implementada por testes de escalonabilidade que podem ou não fazer parte da abordagem de escalonamento adotada. A partir do teste, uma escala de execução é montada, definindo que tarefa utiliza qual recurso a cada momento.

Testes de escalonabilidade podem ser: suficiente mas não necessário, necessário mas

não suficiente e exato. Todo conjunto de tarefa aprovado por um teste suficiente mas não necessário é escalonável mas nada pode ser dito sobre o conjunto reprovado. O conjunto de tarefas reprovado pelo teste necessário mas não suficiente não são escalonáveis, porém, nada pode ser dito sobre o conjunto aprovado. Testes exatos, por outro lado, aprovam conjuntos escalonáveis e reprovam conjuntos não escalonáveis [8].

Diferentes abordagens de escalonamento são identificadas na literatura tomando como base de classificação, aspectos como previsibilidade, utilização de recursos e algoritmos de escalonamento empregados.

A carga a ser tratada pelos recursos computacionais em um determinado instante é definida pelo somatório dos tempos de computação do conjunto de tarefas que compõe a aplicação de tempo real. Quando as tarefas são bem conhecidas em tempo de projeto (restrições temporais conhecidas no tempo de chegada) a carga é caracterizada como estática ou limitada. Por outro lado, quando as características de chegada das tarefas não podem ser antecipadas a carga é dita dinâmica ou ilimitada [10].

Quando a carga é estática, testes de escalonabilidade podem ser aplicados e assim garantir o cumprimento de restrições temporais do conjunto de tarefas em tempo de projeto. Sob estas circunstâncias, caso seja necessário, o redimensionamento do sistema pode ser realizado de acordo com a situação de pior caso [10].

A partir do conhecimento do tempo de computação e período da tarefa, é possível o cálculo da utilização do processador referente a esta tarefa. A utilização reflete a fração do tempo total do processador que a tarefa irá ocupar no pior caso. Para um conjunto de tarefas portanto, o cálculo inclui a somatória de utilização de todas as tarefas do conjunto. Algumas soluções de escalonamento utilizam este conceito.

Soluções de escalonamento estabelecem diferentes critérios ou regras e se baseiam em diferentes aspectos do conjunto de tarefas a ser escalonado. Quando tarefas em execução podem ser preemptadas por tarefas de maior prioridade, o algoritmo de escalonamento é dito preemptivo. Caso contrário, o algoritmo é denominado não preemptivo.

Algoritmos que realizam o cálculo da escala baseado em parâmetros obtidos pelas tarefas de projeto são chamados de algoritmos estáticos. Por outro lado, aqueles que baseiam seu cálculo em parâmetros mutáveis conforme a evolução do sistema, são chamados de dinâmicos.

Os algoritmos que produzem a escala em tempo de projeto são identificados como *offline*, e aqueles que produzem a escala em tempo de execução são algoritmos *online*.

Pela combinação das propriedades apresentadas pelos algoritmos acima descritos, tem-se algoritmos *offline* estáticos, *online* estáticos, *online* dinâmicos.

2.5 Abordagens de Escalonamento

O escalonamento de tempo real pode ser dividido em teste de escalonabilidade e cálculo da escala de execução. O teste de escalonabilidade determina se as restrições temporais do conjunto de tarefas serão atendidas, considerando o algoritmo de escalonamento implementado.

Baseado no tipo de carga tratado e nas etapas de escalonamento, três grupos principais de abordagens podem ser descritos: abordagens com garantia em tempo de projeto, abordagens com garantia em tempo de execução e abordagens de melhor esforço [21].

As abordagens de garantia em tempo de projeto têm como objetivo a previsibilidade determinista, e parte de um conjunto de premissas:

- A carga computacional é conhecida em tempo de projeto (carga estática);
- Os recursos no sistema são suficientes para cumprir tarefas respeitando suas restrições temporais na condição de pior caso.

Estas abordagens são dirigidas para aplicações críticas deterministas. Os dois tipos de abordagens desta categoria são: o executivo cíclico e os escalonadores dirigidos a prioridades.

No executivo cíclico, tanto o teste de escalonabilidade quanto a produção da escala são realizados em tempo de projeto. A escala produzida é executada ciclicamente pelo processador em tempo de projeto. Como a escala reflete o pior caso, embora exista a garantia de cumprimento de restrições, existe também o desperdício de recursos uma vez que o pior caso é geralmente distante do caso médio [10].

Abordagens baseadas em escalonadores dirigidos a prioridade são mais flexíveis, eles realizam o teste de escalonabilidade em tempo de projeto, mas produzem a escala em tempo de execução, por um escalonador dirigido a prioridades. Como o pior caso é considerado somente no teste de escalonabilidade, a solução implementada é mais flexível e ainda assim garante recursos para o pior caso. Exemplos de propostas que seguem esta abordagem podem ser encontrados em [1] e [14].

Esta abordagem faz o uso de escalonador preemptivo que constrói a escala conforme tarefas vão ficando prontas e de acordo com a prioridade das mesmas.

As prioridades são definidas segundo políticas de escalonamento que envolvem atributos estáticos ou dinâmicos (escalonadores *online* estáticos ou dinâmicos).

Quando a atribuição de prioridades é feita estaticamente, as tarefas recebem valores de prioridades durante o teste de escalonabilidade conforme critérios pré-determinados. A política Taxa Monotônica (*Rate Monotonic*) é um exemplo clássico dessas abordagens. Nela, a distribuição de prioridades é feita de maneira inversamente proporcional aos períodos das mesmas [19].

Na atribuição dinâmica de prioridades, a prioridade atribuída à tarefa é alterada pelo escalonador de acordo com a situação temporal corrente da mesma. O escalonador realiza a alteração de prioridades das tarefas considerando outros atributos das mesmas e em momentos pré-determinados pelo algoritmo. Um exemplo clássico desta abordagem é o escalonamento EDF (*Earliest Deadline First*) [17] em que prioridades são atribuídas às tarefas na fila de pronto a cada liberação, conforme a proximidade de cada um de seus deadlines absolutos.

Estendendo o modelo de tarefas do taxa monotônica, o escalonamento *Deadline* Monotônico(DM) [16], assume *deadlines* relativos menores ou iguais ao períodos das tarefas. As demais premissas permanecem as mesmas da abordagem Taxa Monotônica: tarefas periódicas e independentes, tempo de computação constante e tempo de chaveamento entre tarefas assumido como nulo. No *Deadline* Monotônico a atribuição de prioridades é estática e também é dita ótima para sua classe de problemas.

A premissa de tarefas independentes é bastante restritiva, grande parte das aplicações de tempo real determina alguma forma de relação de exclusão e compartilhamento de recursos entre tarefas. Tais relações determinam bloqueios em tarefas mais prioritárias que são identificados na literatura de tempo real como inversões de prioridades.

A inversão de prioridades ocorre em situações em que tarefas compartilham um recurso guardado por um mecanismo de exclusão mútua. Uma tarefa de maior prioridade, T1, é impedida de prosseguir a execução por estar bloqueada, a espera de um recurso sendo usado por uma tarefa de menor prioridade, T4.

Em meio a tal situações, eventuais interferências podem ser provocadas sobre a tarefa de mais alta prioridade (T1). Tarefas de prioridade intermediária (T2 ou T3 com valor de prioridade entre o valor de prioridade de T1 e T4) podem preemptar a tarefa T4 e assim, a tarefa T1 deve esperar que T4 libere o recurso, mesmo estando esta tarefa a espera do processador, sendo ocupado pelas tarefas T2 ou T3. Na situação ilustrada, a tarefa de maior

prioridade do conjunto considerado, T1, permanece indefinidamente a espera da execução de tarefas de menor prioridade.

Uma vez que inversões de prioridade são inevitáveis quando as tarefas apresentam independência, procura-se limitar as eventuais inversões nas escalas produzidas. Assim, alguns métodos para controlar o acesso a recursos compartilhados implementam, através de regras que permitem o conhecimento a priori do pior caso de bloqueio experimentado por uma tarefa no acesso a uma variável compartilhada [10].

Duas técnicas utilizadas para este propósito são: Protocolo de Herança de Prioridade [13] e Protocolo de Prioridade Teto (*Priority Ceiling Protocol*), desenvolvido para esquemas de prioridade fixa. Uma técnica voltada para escalonadores de prioridade dinâmica é a técnica Política de Pilha (*Stack Resource Policy*) [2]. Essas técnicas são descritas em maior detalhes em [10].

Na abordagem de garantia dinâmica (em tempo de execução) tratam da carga computacional não previsível, tarefas com tempo de chegada não conhecidos previamente. Não havendo antecipação de situações de pior caso, não é possível prever recursos para todas as tarefas e nem o cumprimento de requisitos temporais das mesmas. As situações são tratadas em tempo de execução, sendo que a carga dinâmica e os recursos podem ser insuficientes para os cenários de tarefas que se apresentam. Nestas abordagens, tanto o teste de escalonabilidade quanto a produção da escala são feitas em tempo de execução (escalonadores *online* dinâmicos) .

Este grupo de abordagens faz o uso de testes de aceitação a cada nova tarefa que chega no sistema. O teste se baseia em análises realizadas com hipóteses de pior caso sobre alguns parâmetros temporais. Se o teste indica um conjunto não escalonável, a nova tarefa é descartada. Este mecanismo implementa garantia dinâmica e preserva tarefas previamente garantidas como escalonáveis. Estas abordagens são próprias para aplicações com restrições críticas que operam em ambiente não determinista.

Na abordagem de melhor esforço, nenhuma garantia sobre o atendimento das restrições temporais durante a execução é fornecida em tempo de projeto. A carga envolvida é dinâmica, portanto, desconhecida em tempo de projeto. Essa carga é normalmente modelada por tarefas aperiódicas em que os tempos de chegada são desconhecidos. Tais abordagens implicam na eventual ocorrência de sobrecargas transientes, caracterizadas pela ocorrência de recursos disponíveis insuficientes em determinados instantes para as tarefas liberadas para execução.

2.6 Escalonamento de Tarefas Periódicas com Taxa Monotônica

O escalonamento de tarefas periódicas é discutido neste trabalho em esquemas dirigidos a prioridades. Nestes esquemas, prioridades atribuídas às tarefas do conjunto são derivadas de suas restrições temporais e não de atributos como importância ou grau de confiabilidade das tarefas [10].

O algoritmo de prioridade fixa Taxa Monotônica é considerado um algoritmo clássico, dito ótimo para sua classe de problemas e caracterizado por um modelo de tarefas simplificado [17].

Na abordagem Taxa Monotônica a escala é produzida em tempo de execução através de escalonadores preemptivos dirigidos a prioridades. De acordo com as classificações apresentadas, trata-se de um escalonamento estático, *online*. É dito ótimo em sua classe pois nenhum outro algoritmo da mesma classe pode escalonar um conjunto de tarefas não escalonável pelo Taxa Monotônica.

As premissas deste algoritmo definem um modelo de tarefas bastante simples que facilita as análises de escalonabilidade:

- As tarefas são periódicas e independentes;
- O deadline de cada tarefa coincide com seu período (D=P);
- O tempo de Computação (C) de cada tarefa é conhecido e constante;
- O tempo de chaveamento de tarefas é assumido como nulo.

A política que define a atribuição de prioridades usando RM determina uma ordenação baseada nos valores de períodos das tarefas do conjunto; quanto menor o período (maior a freqüência), maior a prioridade.

A análise de escalonabilidade é feita em tempo de projeto e é baseada no cálculo da utilização de processador (U). O teste define condição suficiente para o atendimento de n tarefas:

$$U = \sum_{i}^{n} C_i/P_i \leq n(2^{1/n} - 1) \tag{2.1}$$

A medida que n cresce, a utilização do processador converge para 0.69. Embora uma utilização de 70% defina baixa ocupação do processador, implica no descarte de muitos conjuntos de tarefas com utilização maior e que ainda assim, apresentam escalas realizáveis.

Porém, quando as tarefas do conjunto apresentam períodos múltiplos da tarefa mais prioritária, esta utilização pode ser relaxada para uma condição próxima ao máximo teórico, coincidindo o teste abaixo com uma condição necessária e suficiente[Kop92c]:

$$U = \sum_{i}^{n} C_i/P_i \leq 1 \tag{2.2}$$

Na prática a abordagem de Taxa Monotônica pode escalonar conjuntos de tarefas com utilização mais alta que 0.693. Não é incomum encontrar grandes conjuntos de tarefas periódicas com utilização em torno de 0.90 escalonáveis pelo algoritmo Taxa Monotônica, o que sugere que o comportamento do caso médio é bastante distante do comportamento de pior caso. De fato, o comportamento apresentado pelo conjunto é fortemente dependente dos valores de períodos das tarefas que fazem parte do conjunto[15].

Existem outros modelos de escalonamento mais flexíveis com respectivos testes de escalonabilidade. Em [10] são apresentados uma série de modelos de escalonamento e respectivos testes de escalonabilidade. Os testes de escalonabilidade objetivam determinar se um conjunto de tarefas pode ser escalonado pelo modelo de tarefas adotado mas nem sempre é utilizado em conjunto com o modelo de tarefas.

2.7 Conclusão

Este capítulo apresentou uma breve revisão bibliográfica sobre aspectos principais de tempo real; conceitos, problemática, modelo de tarefas, e abordagens de escalonamento de tempo real.

Devido a grande aplicabilidade de sistemas desta natureza, diversos tipos de sistemas de tempo real podem ser encontrados e várias classificações podem ser atribuídas aos mesmos levando em consideração diferentes aspectos.

O escalonamento de tempo real constitui uma das principais áreas de pesquisa no tema, e caracteriza um problema de grande impacto no comportamento temporal das tarefas tratadas. Assim, baseados em diferentes necessidades e propriedades de aplicações de tempo real, abordagens de escalonamento apresentam técnicas diversas de atender às restrições temporais impostas pelos diferentes tipos de carga encontrados.

Capítulo 3

Linux

O Linux[5] é um sistema operacional *Unix-like* criado no início dos anos 90 por Linus Torvalds na Universidade de Helsinki, Finlandia. Baseado no Minix (pequeno sistema Unix)[26], o Linux foi criado com o objetivo inicial de superar o sistema que o inspirou e acabou se tornando um dos membros mais conhecidos dos sistemas Unix.

Desenvolvido sob a GNU *General Public Licence*, tem seu código fonte aberto, o que garantiu ao mesmo tempo, sua popularidade entre programadores de todo o mundo e sua constante evolução. Atualmente, o kernel se encontra em sua versão 2.4.18, com previsão de funcionalidades voltadas para tempo real na versão 2.5.

O Linux é um kernel convencional, monolítico, não preemptivo com suporte a SMP (*Symmetric Multiprocessor*). Entre outras propriedades, estão inclusas, multiprogramação, memória virtual, biblioteca compartilhada, protocolo de rede TCP/IP e muitas outras características consistentes com um sistema multiusuário tipo Unix[9].

Tendo em vista a grande quantidade de informações a respeito deste kernel, serão abordados neste capítulo somente alguns aspectos do mesmo. A cada nova versão do kernel novas modificações são inseridas, a próxima versão deverá trazer modificações voltadas para melhoria de comportamento de tempo real.

Mesmo não tendo sido desenvolvido com preocupações temporais, a disponibilidade do código oferece uma flexibilidade que tem atraído um número cada vez maior de pesquisadores e projetistas, encorajando o uso deste sistema em aplicações embutidas e de tempo real. Este sistema tem sido adotado há algum tempo para aplicações de tempo real brando em função de seu custo e flexibilidade.

3.1 Interrupções e Exceções

De modo geral, interrupções promovem o desvio do fluxo normal de execução do processador. Na ocorrência de uma interrupção, o processador deve parar o que estiver fazendo e fazer uma troca de contexto para tratar da interrupção ocorrida. O mecanismo é similar a troca de contexto realizada durante o escalonamento de processos, porém o código do manipulador de interrupção ou exceção é um fluxo de controle, que executa no espaço de endereçamento do processo que solicitou o serviço, e é mais leve que um processo.

A manipulação de interrupções é uma das tarefas mais delicadas a ser realizada pelo kernel, pois deve considerar uma série de fatores:

- Interrupções podem ocorrer a qualquer momento, inclusive, em meio ao tratamento de outra atividade do kernel. Para que a interrupção interfira o mínimo possível sobre a atividade corrente e permita que o kernel retome suas atividades, o kernel deve tomar conhecimento da interrupção, mas adiar o máximo possível o seu processamento. Pode-se-ia simplesmente marcar a interrupção como reconhecida e processá-la mais tarde. Este comportamento caracteriza uma divisão no atendimento da interrupção em duas partes denominadas *top-half* e *bottom-half* (substituídas por tasklets na versão 2.4). As *bottom-halves* são representadas por funções em uma fila, a espera para serem executadas num momento posterior.

- Tendo em vista que interrupções podem ocorrer a qualquer momento, o kernel deve estar preparado para a ocorrência de uma interrupção em meio ao tratamento de uma interrupção anterior (interrupção de tipo diferente). Assim, deve existir o suporte a interrupções aninhadas.

- Apesar de aceitar a chegada de novas interrupções em meio ao tratamento de uma interrupção anterior, é preciso proteger seções críticas no código do kernel, desabilitando interrupções. Entretanto, o kernel deve passar o mínimo de tempo possível com interrupções desabilitadas, para que possa estar sempre atendendo novas interrupções, e assim, garantir que dispositivos não sejam mantidos bloqueados [5].

Um fluxo de controle do kernel é uma seqüência de instruções executadas no modo kernel para manipular interrupções e exceções. Embora o projeto do Linux não permita que trocas de contexto ocorram enquanto o processador executa um fluxo de controle do kernel associado a uma interrupção, tais fluxos de controle podem ser arbitrariamente aninhados. Um manipulador de interrupções pode ser interrompido por outro manipulador e assim por diante.

O Linux intercala fluxos de controle do kernel por duas razões principais;

- Melhorar o desempenho de controladores de interrupção programáveis e controladores de dispositivos;
- Implementar um modelo de interrupções sem nível de prioridade. Uma vez que manipuladores podem ser diferidos por outros manipuladores, não há necessidade de estipular prioridades entre dispositivos de hardware.

As exceções no Linux são utilizadas para alcançar duas finalidades:

- enviar sinais a processos notificando condições anômalas (ex: divisão por zero);
- lidar com falta de páginas.

A estrutura de manipuladores de interrupções por outro lado, deve oferecer suporte a ações mais complexas. O manipulador de interrupção deve ser flexível o suficiente para servir vários dispositivos. Várias rotinas de serviço de interrupção são associadas ao mesmo manipulador, cada uma delas é uma função relacionada a um único dispositivo que partilha a mesma linha de IRQ. Como não é possível saber qual dispositivo lançou a IRQ, cada rotina de serviço de interrupção é executada para identificar a origem da interrupção. Uma vez identificada, a rotina de serviço de interrupção correspondente é executada.

Nem todas as ações a serem executadas a partir da ocorrência de uma interrupção precisam ser executadas com urgência. Enquanto o manipulador de interrupções executa, os sinais (eventualmente gerados) correspondentes à mesma IRQ são ignorados. O processo responsável pela interrupção deve estar em num estado apropriado durante o atendimento à interrupção gerada sob pena de provocar a paralização do sistema. Estes fatores levam a concluir que longas operações não críticas devem ser diferidas [5].

Todos os manipuladores de interrupções executam as seguintes funções básicas:

- Salva o valor da IRQ e o conteúdo de registradores na pilha do modo kernel;
- Envia um sinal de reconhecimento ao controlador de interrupção programável que fornece serviços à IRQ;
- Termina, efetuando um salto para o endereço da função *ret_from_int()*.

Embora fluxos de controle do kernel possam ocorrer sem interferência, isto é, iniciar e concluir em uma única seqüência, existem situações em que intercalação de fluxos de controle de kernel ocorrem. Uma destas situações é a própria troca de contexto, uma outra é

a ocorrência de interrupção enquanto o kernel executa um fluxo de controle com interrupções habilitadas. Neste caso, o primeiro fluxo de controle é deixado inacabado enquanto o processador executa o tratador de interrupção.

Uma vez permitindo a intercalação de fluxo de controle do kernel, cuidados especiais em relação a estruturas de dados devem ser tomadas a fim de que não haja perigo de corrupção de dados.

Quatro técnicas de sincronização se aplicam a fluxos de controle do kernel.

- não preempção de processos em modo kernel;
- operações atômicas;
- interrupções desabilitadas;
- locks.

A primeira delas determina que processos em execução no modo kernel só podem ser substituídos quando liberam voluntariamente o processador. Além disso, a manipulação de interrupções e exceções pode interromper processos executando em modo kernel, porém, ao término dos mesmos, o fluxo de controle do kernel é reassumido. E que fluxos de controle do kernel podem ser interrompidos somente por outros fluxos de controle que executam a manipulação de interrupções e execuções. Com isso, os fluxos de controle do kernel que lidam com chamadas não bloqueantes são atômicas em relação a outros fluxos de controle iniciados por chamadas do sistema.

Operações atômicas garantem atomicidade através de instruções assembly. Além de instruções simples, uma série de instruções um pouco mais sofisticadas nesta linguagem são disponibilizadas para efetuar operações de forma atômica.

Porém, trechos de código extensos não podem ser implementados por instruções atômicas, exigindo mecanismos mais sofisticados para assegurar a não ocorrência de condições de corrida. A execução com interrupções desabilitadas fornece esse mecanismo e permite proteger efetivamente as estruturas de dados até mesmo de sinais de IRQ de dispositivos de hardware. Ainda assim, desabilitar interrupções não previne intercalação de fluxos de controle do kernel. Uma exceção de falta de página poderia ser gerada suspendendo o processo corrente.

Por uma razão de simplicidade, o mecanismo de interrupções desabilitadas é amplamente usado por funções do kernel na implementação de seções críticas. Entretanto, a região protegida por este mecanismo deve ser pequena, pois bloqueia a comunicação entre processador

e dispositivos de E/S. A técnica de *lock* é aplicada a regiões críticas longas, e funciona de modo a permitir somente um fluxo de controle na região de cada vez. Ela é implementada através de semáforos: quando um fluxo de controle do kernel tenta obter um recurso em *lock*, o processo correspondente a este fluxo é suspenso e só volta a executar novamente quando o recurso é liberado.

3.2 O tempo no Linux

No Linux sobre PC-Pentium, existem três mecanismos que interagem com o kernel e são voltados para o registro de passagem no tempo: o *Real Time Clock* (RTC), o *Time Stamp Counter* (TSC) e o *Programmable Interrupt Timer* (PIT).

O relógio de tempo real (RTC) e o contador de *timestamps* (TSC) são dispositivos de hardware que permitem ao kernel acompanhar o tempo corrente do dia, e o último é programado pelo kernel de forma que gere interrupções a uma freqüência fixa e predefinida. Estas interrupções são cruciais para implementação de temporizadores usados pelo kernel e programas de usuários [BOVET].

O relógio de tempo real é independente do processador e está presente em todo computador pessoal. Ele permanece em funcionamento mesmo quando a máquina é desligada. O Linux usa o relógio de tempo real somente para verificação de horário e data, porém, sua programação é permitida através de arquivos de dispositivos (/dev/rtc).

O contador de *timestamps* é um registrador de 64 bits presente nos microprocessadores Intel 80x86 a partir do Pentium que pode ser lido através da instrução `assembly rdtsc`. Este registrador é um contador incrementado a cada sinal de *clock*. Ele recebe sinais do *clock* de um oscilador externo e provê uma medida de tempo muito mais exata que o temporizador programável de interrupções, a partir da freqüência de sinal de *clock* calculada no momento de inicialização do kernel.

O temporizador programável de interrupções, por sua vez, tem como objetivo principal o cálculo de um intervalo de tempo. Ele é responsável por gerar as chamadas interrupções de temporizador (`timer`) que notifica o kernel que um novo intervalo de tempo se passou. As interrupções são geradas indefinidamente, em uma freqüência fixa, estabelecida pelo kernel.

Em arquiteturas PC convencionais, existe ao menos um temporizador programável de interrupções, e o primeiro deles é programado pelo Linux para gerar uma interrupção de `timer` na IRQ 0 a uma freqüência de 100Hz, ou seja, a cada 10ms. Este intervalo de tempo é chamado de *tick* de *clock*, e determina o rítmo de todas as atividades do sistema.

Aumentar a freqüência com que as interrupções de `timer` são geradas, aumentaria a resolução do sistema. Porém, isso faz com que o sistema gaste mais tempo executando em modo kernel e menos em modo usuário, tendo em vista o aumento do número de interrupções de *timer* geradas e que devem ser tratadas. Conseqüentemente, programas no espaço do usuário executariam mais lentamente, por ter menor tempo de processador dedicado a eles. Por isso, somente máquinas com processadores bastante velozes podem adotar maior freqüencia de *ticks* sem tornar o sistema perceptivelmente mais lento[22].

Os eventos do Linux sofrem grande influência do temporizador e das interrupções geradas por ele.

A freqüência do *timer* é determinada pelas seguintes macros:

- HZ: armazena o número de interrupções do *timer* por segundo, isto é, a freqüência de interrupções do temporizador. Para os PCs e compatíveis e para a maior parte das outras plataformas, este valor é definido em 100Hz.
- CLOCK_TICK_RATE: armazena o valor 1193180, que é a freqüência do oscilador interno do chip 8254.
- LATCH: armazena a taxa entre CLOCK_TICK_RATE e Hz. E é usado para programar o PIT.

A inicialização do primeiro PIT é realizada com uma pequena seqüência de instruções que carregam o PIT com o valor da taxa armazenada em LATCH (1 instrução para que ao PIT gere interrupções a uma nova taxa, duas instruções para carregar o conteúdo de LATCH - 16bits na porta de E/S-8bits do dispositivo).

A cada ocorrência de interrupção de `timer` um conjunto de atividades é disparado. As principais delas são: atualização do tempo desde a inicialização, atualização de tempo e data, determinação da quantidade de tempo que o processo esteve executando no processador e conseqüente preempção do mesmo caso tenha excedido o tempo alocado, atualização de temporizador de software fazendo chamada de funções apropriadas nos casos em que o intervalo tenha sido cumprido.

A atualização do tempo desde a inicialização do sistema e a atualização de data e hora são consideradas atividades urgentes, e portanto, executadas pelo próprio manipulador de interrupções. As demais atividades são realizadas pelos *bottom halves* do temporizador.

O kernel mantém duas funções para manutenção do tempo, uma para manter atualizada data e hora, e outra para contar o número de microssegundos transcorridos dentro do segundo

corrente. A manutenção destes valores é feita por funções de armazenamento em variáveis do kernel que apontam diretamente para funções de leitura do contador de *timestamps*.

O tempo corrente é calculado por `do_gettimeofday()` que é apontada por um ponteiro de variável `do_get_fast_time`. O número de microssegundos é calculado por `do_fast_gettimeoffset()`, cujo endereço é armazenado em `do_gettimeoffset()`.

Uma vez inicializada a porta da interrupção IRQ0, o campo handler do descritor `irqaction` da IRQ0 contém o endereço da função `timer_interrupt()`. Esta função começa a execução com interrupções desabilitadas, uma vez que o campo `status` do descritor principal de IRQ0 tem seu *flag* SA_INTERRUPT ligado.

Então, os seguintes passos são executados:

- Executa uma instrução assembly `rdtsc` para armazenar o valor do registrador TSC na variável `last_tsc_low`;
- Lê o estado do oscilador interno do chip de dispositivo 8254 e processa o atraso entre a ocorrência de interrupção de temporizador e a execução da rotina de serviço de interrupção;
- Invoca `do_timer_interrupt()`. Esta função pode ser considerada como a rotina de serviço de interrupção comum a todos os modelos 80x86 e executa as operações:
  - invoca a função `do_timer()`;
  - se uma chamada `adjtimex()` (chamada para alteração de hora) foi disparada, invoca a função `set_rtc_mmc()` a cada 660 segundos (11 minutos) para ajustar o relógio de tempo real.

A função `do_timer()`, que executa com interrupções desabilitadas, deve ser executada o mais rapidamente possível. Por isso, simplesmente atualiza um valor fundamental, o tempo decorrido desde a inicialização do sistema, enquanto atribui a todas as demais atividades às duas `bottom_halves`. Esta função faz referência a três variáveis principais relacionadas a medições de tempo, a primeira é a de atualização do valor fundamental recém mencionada, enquanto as outras duas são necessárias para armazenar *ticks* perdidos que ocorrem antes que as funções `bottom_halves` tenham a chance de executar.

Assim, a primeira é absoluta (sempre sendo incrementada) enquanto as outras duas são relativas a outra variável, chamada `xtime` que armazena uma aproximação do tempo corrente. As variáveis de `do_timer()` são:

- `jiffies`: O número de *ticks* decorridos desde a inicialização do sistema, iniciado com 0 durante a inicialização e incrementado quando a interrupção de *timer* ocorre, a cada *tick*.

- `lost_ticks`: O número de *ticks* ocorridos desde a última atualização de `xtime`.

As funções de `bottom_halves` (`time_bh()` e `TIMER_BH`) invocam as funções auxiliares `update_times()`, `run_old_timers()` e `run_timer_list()`.

A função `update_times()` faz a atualização da variável `xtimes`, que é a variável usada por programas de usuários para leitura de data e hora atuais. A atualização é feita com interrupções desabilitadas e baseada no valor indicado por `lost_ticks`.

A função `run_old_timers()` é usada na atualização de estatística de uso de recurso e a função `run_timer_list()` é usada para atualizar listas de temporizadores dinâmicos.

As interrupções de tempo no linux tem um papel crucial principalmente no que diz respeito ao escalonamenteo de tarefas. Uma vez que o Linux utiliza um algoritmo de escalonamento baseado em fatias de tempo(*time-sharing*), o tempo durante o qual cada processo permanece em execução no processador é contabilizado nas atualizações das variáveis manipuladas pelas interrupções de tempo. O *quantum* (quantidade de tempo o qual é permitido a cada processo que permaneça no processador) é sempre múltiplo de um *tick* (10ms). E o tempo durante o qual o processo permanece em execução é indicado pelo campo `counter` do descritor de cada processo que é tambem atualizado por um *bottom_half*, portanto, de em um momento diferente.

Temporizadores são amplamente utilizados tanto pelo kernel quanto por processos de usuários e são facilidades de software que permitem a invocação de funções em um momento futuro ou a detecção de condições anormais de dispositivos, por exemplo. O Linux disponibiliza três tipos de temporizador; estáticos, dinâmicos e de intervalo. Os dois primeiros são usados pelo kernel, enquanto o segundo pode ser criado por processos em modo usuário.

É importante lembrar que tais mecanismos também estão sujeitos a atrasos, uma vez que as funções referentes a estes são também realizados por *bottom_halves* e podem ser executadas um longo tempo após serem ativados [5].

3.3 Chamadas de Sistema

As chamadas de sistema são o meio pelo qual o Linux e demais sistemas Unix-like implementam a interface entre processos do modo usuário e dispositivos de hardware. Embora

esta camada extra entre hardware e aplicações forneça ao sistema facilidades de programação, portabilidade e segurança, são também fonte de não determinismo, tendo em vista se tratar de mais um passo intermediário entre requisição e realização de um serviço para o qual não existe medida exata de duração de tempo.

Quando um processo de modo usuário realiza uma chamada de sistema, o processador troca o contexto para modo kernel e inicia a execução da requisição. O número da chamada de sistema é passado como parâmetro para identificar a chamada requerida que retorna sucesso ou falha.

O manipulador da chamada de sistema executa os seguintes passos:

- Salva o contexto da maioria dos registradores na pilha do modo kernel;
- Trata a chamada de sistema invocando a função correspondente, denominada rotina de serviço de chamada de sistema;
- Sai do manipulador através da chamada da função `ret_from_sys_call()` [5].

3.4 Escalonamento

Assim como os demais algoritmos implementados em sistemas do tipo Unix, o algoritmo de escalonamento do Linux objetiva munir o sistema com tempo de resposta rápido, bom desempenho de tarefas em *background*, evitar que processos sofram postergação indefinida, reconciliar as necessidades de processos de alta e baixa prioridade entre outros.

O conjunto de regras adotado para determinação de quando e como selecionar o processo a ser executado é denominado política de escalonamento.

O escalonamento no Linux é baseado na técnica de fatias de tempo. Nela, vários processos podem executar concorrentemente (ou pseudo-concorrentemente) fazendo uso de fatias de tempo do processador (*quantum*). A técnica de fatias de tempo se baseia em interrupções de tempo. Os aspectos relacionados a mecanismos de escalonamento são transparentes para os processos.

Esta política baseia-se também na ordem de prioridade dos processos. No Linux a prioridade de processos é dinâmica, o sistema operacional ajusta periodicamente a prioridade de cada processo. Assim, processos que há muito não utilizam o processador têm sua prioridade aumentada enquanto os processos freqüentemente executados sofrem o processo inverso.

De modo geral, processos podem ser classificados em *CPU bound* ou *I/O bound*. Tais processos correspondem a processos de uso intensivo de processador ou uso intensivo de dispositivos de E/S respectivamente. Uma vez que processos a mais tempo sem usar o processador tem seus valores de prioridade incrementados, existe implicitamente o favorecimento de processos *I/O bound* ou de processos com prioridade muito baixa.

Uma outra classe de processos no Linux são os processos de tempo real, que são explicitamente identificados e recebem maior prioridade sobre os demais processos. Assim como os demais processos do Linux, os processos de tempo real, ao contrário do kernel, são preemptivos. No momento em que um processo mais prioritário é inserido na lista `runqueue`, o processo corrente do processador é interrompido e o escalonador é chamado para selecionar outro processo. Os processos que não são escalonados por SCHED_FIFO podem ainda ser interrompidos ao final de seu *quantum*.

A duração do *quantum* possui grande impacto sobre o desempenho do sistema. Não deve ser longo, sob pena de sacrificar a velocidade de resposta do sistema a ponto de não mais parecer um sistema concorrente. E ao mesmo tempo não pode ser demasiadamente pequeno pois provocaria a sobrecarga do sistema com atividades relacionadas com troca de contexto de processos.

O algoritmo de escalonamento do Linux divide o tempo do processador em épocas. No início de cada época é contabilizado *quanta* de execução de todas as tarefas do sistema, e assim, a época termina quando todos os processos tiverem gasto todo o seu *quantum*, dando início a uma nova época.

Inicialmente, todo processo herda do processo pai, o valor do *quantum*, cujo valor base é definido como 20 *ticks*, ou 210ms, que geralmente é o *quantum* utilizado pela maior parte dos processos do sistema.

A seleção de um novo processo para execução no processador considera dois tipos diferentes de prioridades de cada processo: prioridade estática e prioridade dinâmica. A prioridade estática é usada por processos de tempo real, são definidas pelo usuário e podem assumir valores de 0 a 99.

A prioridade dinâmica se aplica apenas a processos não tempo real, e basicamente é dada pela soma do *quantum* base e o número de *ticks* restantes de processador para uso do processo até que seu *quantum* termine (dentro da época corrente).

Processos não tempo real somente são selecionados para execução na ausência de qualquer processo de tempo real na lista `runqueue`.

Cada descritor de processos inclui diversos campos voltados para escalonamento:

- `need_reesched`: uma *flag* verificada pela função `ret_from_intr()` para decisão de invocar a função `schedule()` ou não. A função `schedule()` é a função que efetivamente implementa o escalonamento no Linux, ela tem como objetivo, encontrar um processo na lista *runqueue* e atribuir a ele o processador.
- `policy`: que indica a classe de escalonamento. As classes permitidas são:
  - SCHED_FIFO: que implementa o algoritmo *first-in-first-out* como critério de desempate no caso de prioridades iguais para processos de tempo real. Quando o escalonador atribui o processador ao processo, ele deixa o descritor do processo em sua posicão corrente na fila de prontos (`runqueue list`). Se não existe outro processo de tempo real de maior prioridade pronto para executar, o processo continua a ocupar o processador enquanto for preciso.
  - SCHED_RR: que implementa o algoritmo *Round-Robin*. Quando o escalonador atribui o processador ao processo, coloca o descritor do processo na lista `runqueue`. Esta política garante justiça na distribuição de tempo de processador entre processos de mesma prioridade no escalonador SCHED_RR.
  - SCHED_OTHER: implementa o escalonador padrão *time-sharing*.

O campo `policy` também codifica a *flag* binária SCHED_YIELD. Esta flag é ligada quando o processo invoca a chamada de sistema *sched_yield()* (uma maneira voluntária de liberar o processador sem que seja necessário a invocação de operações de E/S ou *sleep*). O escalonador coloca o descritor de processos na base da lista `runqueue`.

- `rt_priority`: indica a prioridade estática de processos de tempo real. Este campo não é usado por processos comuns.
- `priority`: o *quantum* base, ou prioridade base do processo.
- `counter`: o número de *ticks* de processador restantes do processo antes que o *quantum* expire; quando uma nova época inicia, este campo contém a duração do tempo em *quantum* dos processos. Este campo é decrementado a cada *tick* pela função `update_process_times()`.

Quando um novo processo é criado, a função `do_fork()` inicializa o campo `counter` dos processos pai e filho. De fato, o número de *ticks* é dividido entre pai e filho, o que previne a criação indefinida de processos como um meio de conseguir mais tempo de processador.

Entre processos convencionais, campos `priority` e `counter` são usados tanto para implementar *time-sharing* quanto para computar prioridade dinâmica. Para processos escalonados por SCHED_FIFO, simplesmente não são utilizados.

A função *schedule()*, como mencionado anteriormente, implementa o escalonador. A invocação desta função pode ser feita diretamente ou através de *lazy invocation* por diversas rotinas do kernel.

A invocação direta é feita quando o processo corrente deve ser bloqueado imediatamente pois o recurso que necessita não está disponível. Nesse caso, a rotina de kernel que deseja bloquear o processo executa os seguintes passos:

- insere `current` (processo apontado por `current`) na *wait queue* correspondente;
- muda o estado de `current` para TASK_INTERRUPTIBLE ou TASK _UNINTERRUPTIBLE;
- invoca `schedule();`
- verifica se o recurso está disponível (caso contrário retorna ao segundo passo);
- uma vez que o recurso estiver disponível, remove `current` da *wait queue*.

Percebe-se nesta seqüência de ações, que o kernel verifica repetidamente se o recurso necessário pelo processo está disponível. Caso não esteja, o processador é atribuído a um segundo processo pela invocação de `schedule()`. Posteriormente, quando o processador é atribuído ao primeiro processo novamente, o recurso é novamente verificado.

O escalonador é também diretamente invocado por diversos *drivers* de dispositivos que executam longas tarefas iterativas. A cada ciclo de iteração, o *driver* verifica o valor do campo `need_reesched` e se necessário, invoca `schedule()` para liberar voluntariamente o processador [5].

A *lazy invocation* é caracterizada pela atribuição do valor 1 ao campo `need_reesched` do processo `current`. Como este campo é sempre verificado antes do término da execução de processos em modo usuário, a função `schedule()` será invocada em algum momento futuro. Este tipo de invocação é feita nos seguintes casos:

- quando o processo corrente exauriu seu *quantum*;
- quando um processo acorda e sua prioridade é mais alta que a prioridade do processo corrente.
- quando uma chamada de sistema `sched_setscheduler()` ou `sched_yield()` é realizada.

Antes de realmente escalonar um processo, a função `schedule()` executa outras funções, deixadas pendentes em várias filas, e então executa *bottom-halves* ativas. Só então o escalonamento realmente é realizado;

O valor de `current` é salvo na variável local `prev` e o campo `need_resched` de `prev` recebe 0. Uma variável `next` aponta para o descritor do processo selecionado para substituir `prev`.

Primeiro, uma verificação determina se `prev` é um processo *Round-Robin* que exauriu seu *quantum*. Caso seja, `schedule()` atribui um novo *quantum* a `prev` e o coloca no fim da lista `runqueue`.

Então, `schedule()` examina o estado de `prev` e caso ele tenha sinais pendentes não bloqueados, e seu estado é TASK_ININTERRUPTIBLE, a função acorda o processo, atribuindo ao mesmo o estado TASK_RUNNING. Isso implica apenas em dar a `prev` a chance de ser selecionado para execução [5].

Se o estado de `prev` não é TASK_RUNNING, então `schedule()` foi invocado diretamente pelo processo que deveria estar a espera de um recurso, por isso, `prev` dever ser removida da `runqueue`.

`schedule()` deve selecionar o processo a ser executado no próximo *quantum* e para isso, a função percorre a lista `runqueue`. Começando pelo processo referenciado pelo campo `next_run` de `init_task`, a função busca armazenar em `next` o ponteiro do descritor de processo referente ao processo de maior prioridade. Para isso, utiliza na seleção a função `goodness()`, que identifica na lista `runqueue` o melhor candidato para execução. Baseado nos campos `policy` e `counter` da função avaliada e em `prev`, `goodness()` retorna um valor inteiro. Quanto maior o valor, maior a chance do processo avaliado ser selecionado para execução. `schedule()` chama repetidamente a função `goodness()` para determinar entre os processos prontos para executar, o melhor candidato.

Caso o processo selecionado seja diferente do processo corrente, a troca de contexto é efetuada. Quando não existem processos na lista `runqueue`, `next` aponta para a função `init_task`. Quando todos os processos tem prioridade menor ou igual a prioridade de `prev`, não há troca de contexto e o processo corrente permanece em execução. Uma verificação posterior ao laço de chamadas a `goodness()` verifica se todos os processos exauriram seu *quantum*, e em caso afirmativo, uma nova época tem início, com a atribuição de novos valores de *quantum* aos processos.

Embora o algoritmo de escalonamento seja simples e com regras bem definidas, ele permanece um componente misterioso no kernel. É possível modificar sua performance de

maneira significativa mudando poucos parâmetros, mas não há evidências teóricas que justifiquem os resultados obtidos. Não há como certificar que os resultados obtidos, bons ou ruins, se repetirão uma vez que o conjunto de requisições feitas por usuários variam significativamente.

Em [5], são listadas algumas propriedades negativas do escalonador implementado pelo Linux:

- O algoritmo é ineficiente para um número grande de processos devido ao recálculo do valor de prioridade dinâmica.
- O valor de *quantum* predefinido é muito alto para cargas altas no sistema. O tempo de resposta do sistema é altamente dependente da quantidade de processos que esperam para executar, e o valor do *quantum* se mostrou inapropriado para situações em que a carga esperada é alta.
- A estratégia de favorecimento de processos *I/O bound* não é uma estratégia ótima. Nem sempre processos que fazem uso intensivo de dispositivos de entrada e saída são processos interativos, podem nem mesmo ser processos com necessidade de tempo de resposta curto, mas acabam sendo favorecidos injustamente pelo escalonador.
- O suporte a aplicações de tempo real é fraco. Apesar dos escalonadores e funções de manipulação de prioridade de processos, as propriedades do Linux são contrárias as características que compõem um sistema operacional de tempo real.

3.5 Aspectos Gerais de Implementação de Aplicações de Tempo Real

Grande parte das aplicações de tempo real estabelecem comunicação com algum tipo de dispositivo de hardware, sendo acionadas por interrupções geradas pelo mesmo. Exemplos comuns de aplicações que se encaixam nesta descrição são aquelas acionadas pela leitura de temperatura, dado ou sensor ou pelo sinal enviado ao pressionar de um botão.

Neste tipo de aplicação, as atividades ocorridas no espaço de endereçamento do kernel, como a performance do manipulador de interrupções (no *driver* de dispositivo de hardware utilizado), são de grande importância para a análise do comportamento da tarefa de tempo real. Porém, não são os únicos fatores de influência sobre o mesmo. O comportamento dos demais drivers de dispositivos, a política adotada pelo kernel em relação a desabilitar interrupções e o modelo de preempção do kernel estão diretamente relacionados ao tempo contabilizado durante a execução da tarefa de tempo real [7].

A análise apresentada em [7] descreve uma seqüência padrão seguida por este tipo de aplicação pode ser descrita:

- o dispositivo relacionado à aplicação gera a interrupção indicando a existência de dados a serem entregues;
- o manipulador de interrupção copia os dados do dispositivo para o espaço do usuário e ativa a tarefa de tempo real;
- o *driver* de dispositivo executa;
- a tarefa executa, processando os dados e notifica uma tarefa de fornecimento de resultados.
- a tarefa por sua vez, fornece os resultados da aplicação a um usuário, ao ambiente, ou a qualquer que seja o destinatário dos dados gerados.

A descrição de eventos envolvidos durante esta seqüência pode ser dividida em etapas sobre o domínio do tempo. A primeira etapa tem início com o disparo da interrupção. A segunda etapa começa no instante do início da execução do manipulador de interrupção, seguida pela etapa que de posicionamento da tarefa de tempo real na fila de *runqueue*. A quarta etapa tem início quando a tarefa efetivamente começa a execução, e é encerrada pelo instante em que a tarefa completa a ativação.

Analisando superficialmente o comportamento temporal desta aplicação hipotética, alguns fatores podem ser apontados. Nas duas primeiras etapas; do momento em que a interrupção é gerada ao momento em que a tarefa é colocada na fila de *runqueue*, o tempo gasto é geralmente mínimo, porém totalmente dependente da maneira com que o manipulador é implementado. Testes realizados sobre estes pontos de aplicações mais comuns mostram que a primeira etapa gasta em média 15us. A segunda etapa (início do manipulador de interrupção à posicionamento da tarefa na fila de *runqueue*), pode ser ainda mais curta, porém é dependente da ocorrência ou não de outras interrupções estarem ocorrendo no sistema. A etapa com início no posicionamento da tarefa na fila de *runqueue* pode ser longa. O tempo gasto nesta etapa depende de atributos definidos para a tarefa e do algoritmo utilizado em seu escalonamento. Finalmente, a última etapa, delimitada pelo início da execução da tarefa e pelo término da ativação, tem um tempo dependente do código da tarefa em sí, e da interferência que pode ser provocada por outras atividades em execução no sistema [7].

Através desta descrição, percebe-se que atrasos inevitáveis são introduzidos em qualquer projeto de aplicação de tempo real. E ainda que, teoricamente, a plataforma forneça mecanismos suficientes para satisfazer os requisitos temporais da aplicação, a construção da mesma

deve ser feita de forma cuidadosa, e considerar também aspectos externos (à aplicação) na verificação de seu comportamento temporal.

Diversos fatores diferentes influenciam sobre o comportamento da aplicação. Cada etapa de seu funcionamento está sujeita a um conjunto em particular destes fatores. A partir do momento em que a interrupção do dispositivo utilizado é gerada, até o instante em que o manipulador de interrupção começa a executar, interrupções desabilitadas, a ocorrência de outras interrupções de hardware, a espera pelo uso de barramento, o tempo gasto para trazer dados e código do manipulador para memória *cache*, o código do kernel, e até mesmo a maneira com que o *driver* de dispositivo é escrito, podem influenciar sobre o tempo gasto durante esta etapa. Na construção da aplicação, o trecho de código com interrupções desabilitadas deve ser o menor possível, e deve-se evitar fazer chamadas ao sistema que possam provocar atividades no kernel no momento em que interrupções do dispositivo da aplicação possam chegar [7].

Outras interrupções de hardware ocorrendo no sistema e o modo como o *driver* de dispositivo é escrito, influenciam também sobre a segunda etapa da ativação, quando o manipulador de interrupção passa a executar. Esta etapa, entretanto, geralmente executa em tempo mínimo em comparação com as demais etapas.

Na terceira etapa da ativação da aplicação de tempo real de acordo com [7], interrupções de hardware de outros dispositivos ainda podem ocasionar um atraso na execução da aplicação. Porém, verifica-se que a ocorrência de interrupções deste tipo é baixa em aplicações de tempo real. Nesta etapa, o fato do kernel do Linux ser não preemptivo e executar com maior prioridade que tarefas no espaço de endereçamento do usuário exercem maior influência sobre o tempo de atraso. Eventualmente, o processador pode estar executando trechos de código do kernel em atendimento a uma chamada de sistema provocada por um processo de prioridade mais baixa que a tarefa de tempo real, provocando o atraso da mesma. Tais atrasos não podem ser delimitados de forma determinista no Linux, o que insere indeterminismo na execução da etapa de ativação da tarefa.

Medições[11] indicam que o Linux pode prover uma latência máxima de 100ms . Questões relacionadas ao escalonamento de tarefas também influenciam sobre o atraso desta etapa da aplicação. Ser implementada com atributos de uma tarefa de tempo real com prioridade alta não garante a preferência do processador em qualquer circunstância pois a otimização do Linux para o atendimento do caso médio pode provocar atrasos em tarefas de prioridade mais altas.

A última etapa da ativação da tarefa de tempo real da aplicação, começa com o início da tarefa de tempo real e deve manter a tarefa executando com o melhor comportamento possível. Técnicas de programação influenciam nesta etapa, além disso, a ocorrência de

interrupções de hardware, a manipulação de memória cache, falta de páginas e execução de *bottom-halves* podem atrasar a tarefa.

O uso de semáforos implica na possibilidade de sujeitar a tarefa de tempo real a um tempo indefinido de bloqueio, caso o mecanismo não implemente algum tipo de prevenção quanto a inversão de prioridades. O uso de *drivers* de dispositivos também influenciam sobre esta etapa. No Linux, o *driver* de disco é construído de maneira a fazer requisições em lote, o que ocasionalmente atrasa as requisições da tarefa.

Portanto, esta etapa exige que a tarefa de tempo real seja construída com boas técnicas de programação, métodos de sincronização que previnam inversão de prioridade quando possível, e *drivers* de dispositivos que ofereçam suporte a tempo real.

3.6 Conclusão

O Linux é um kernel de propósitos gerais cuja construção e mecanismos estão voltados para a obtenção de melhor desempenho no caso médio. O fato de dispor de código aberto, bom desempenho, e oferecer flexibilidade na implementação de *drivers* de dispositivos tem encorajado pesquisadores e projetistas a investir neste sistema como uma solução no suporte a aplicações de áreas diversas, entre elas a área de tempo real brando.

Este capítulo abordou os principais aspectos deste kernel de sistema operacional. Procurou-se abordar as propriedades de maior impacto sobre tempo de resposta deste sistema e também aquelas que envolvem funcionalidades diretamente relacionadas a concorrência (fator importante no tratamento de sistemas de tempo real).

Foram ilustradas característica de tratamento e funcionamento de mecanismos que manipulação de tempo, chamadas de sistemas, escalonamento, interrupções e exceções e sincronização.

A passagem de tempo no Linux é mantida por contadores de *ticks* de clock, osciladores externos independentes de processador e geradores de interrupção. O temporizador de intervalos programável é responsável por gerar as interrupções de tempo que determinam o rítmo de várias atividades no sistema, mantendo um período de 10ms.

Chamadas de sistemas disponibilizam o acesso de aplicações em modo usuário a funcionalidades implementadas dentro do kernel.

O escalonamento é implementado por um algoritmo relativamente simples, mas que apresenta algumas desvantagens. Apesar de disponibizar algoritmos específicos para processos

de tempo real, se mostra ineficiente para atender requisitos de processos desta natureza. Modificações simples podem proporcionar resultados diferentes porém não possuem base teórica para comprovação de melhoria, tão pouco garantem o comportamento similar diante de diferentes conjuntos de carga no sistema.

Fluxos de controle do kernel definem seqüências de atividades executadas no modo kernel em função de processos de usuários ou em decorrência de interrupções ou exceções. Mecanismos de sincronização implementam proteção às seções críticas no kernel, garantindo o mínimo de falhas mesmo diante da propriedade de intercalação de fluxos de controle.

A composição de uma descrição breve sobre aspectos específicos do funcionamento do kernel é dificultada pela complexidade de seu código. Em inúmeras circunstâncias rotinas do kernel se relacionam de forma a dificultar a análise isolada de mecanismos específicos. O sistema completo, incluindo programas de sistema, tem cerca de 14000000 linhas de código. Uma descrição mais detalhada do kernel do Linux pode ser encontrada em [5], [18], [3], [22].

Capítulo 4

Condições das Experiências

Este capítulo objetiva ilustrar aspectos importantes que fazem parte da construção dos cenários cujos experimentos são executados. Serão descritos também alguns dos mecanismos que fazem parte das aplicações. De maneira direta ou indireta, seu comportamento afeta os resultados apresentados no capítulo 5.

No decorrer da etapa de preparação dos experimentos procurou-se basear as aplicações na teoria de tarefas periódicas e no conhecimento existente na literatura a respeito de tarefas de natureza temporal.

Ao mesmo tempo, conclusões resultantes de experiências preliminares foram também consideradas. Os aspectos relevantes relacionados ao conhecimento aplicado à construção dos experimentos deste trabalho serão documentados com maior riqueza de detalhes no decorrer deste capítulo.

A seção 4.1 descreve genericamente o processo de implementação de tarefas periódicas no Linux. São ilustrados os principais aspectos relacionados tanto às dificuldades encontradas, quanto ao conhecimento adquirido durante o processo. A seção 4.2 detalha a implementação das tarefas utilizadas no trabalho, nomeando funções e mecanismos utilizados. Nesta seção são detalhadas as decisões tomadas durante a construção das aplicações mediante testes executados com as mesmas. Na seção 4.3 são caracterizados os cenários nos quais cada aplicação foi executada. Finalmente a seção 4.4 traz conclusões a respeito dos tópicos abordados neste capítulo.

4.1 Tarefas Periódicas no Linux

Sistemas de tempo real, geralmente, são descritos através de um modelo de execução baseado em tarefas (*task*) [8]. O modelo de tarefas descrito no capítulo 2 define um conjunto de restrições temporais que permite a determinação do comportamento da tarefa no tempo. A implementação de tarefas de tempo real se baseia nos conceitos do modelo. Porém, nem todas as restrições definidas no mesmo podem ser estabelecidas durante a implementação ou mesmo reconhecidas durante a execução destas tarefas. Neste trabalho, o termo tarefa é usado no sentido abstrato, independente de sistema operacional.

Tradicionalmente, tarefas de tempo real são implementadas por *threads*. *threads* são programas em execução com o objetivo principal de promover concorrência e são caracterizadas pela capacidade de permitir que múltiplas tarefas executem simultaneamente com o mínimo custo possível sobre o sistema operacional [4]. Estas propriedades contribuem para a escolha deste recurso na implementação de tarefas de tempo real. Por isso este trabalho faz o uso de *threads* e processos na construção das tarefas.

No Linux, aspectos de implementação possuem grande influência sobre o comportamento das tarefas. Propriedades inerentes ao sistema tornam tarefas que executam sobre ele imprevisíveis. O fato de constituir um sistema operacional de propósitos gerais não o torna simplesmente não determinista, mas também impõe uma série de restrições quanto aos métodos de programação da tarefa e quanto ao desempenho da mesma.

Mesmo sendo projetado para aplicações de propósitos gerais, o Linux disponibiliza alguns mecanismos voltados para melhoria de desempenho de tarefas de tempo real. Um deles é a possibilidade de escolha entre algoritmos de escalonamento de tempo real. São disponibilizados dois escalonadores, SCHED_FIFO e SCHED_RR. Estes algoritmos implementam as políticas *First-in-First-out* e *Round-Robin* respectivamente como um critério de desempate no escalonamento de tarefas de mesma prioridade. A escolha do escalonador de tempo real permite atribuir, de acordo com o mesmo, a prioridade das *threads* criadas. O funcionamento destes escalonadores e também do tratamento de *threads* implementadas no Linux estão de acordo com o padrão POSIX.

Além destes recursos, que podem ser apenas considerados como um esforço na busca por melhor desempenho, não existe no Linux padrão qualquer API (*Aplication Program Interface*) que implemente funcionalidades para tornar *threads* periódicas, por exemplo.

A construção de tarefas periódicas no Linux depende unicamente do esforço do programador. Esta atividade é realizada pela combinação de recursos disponíveis no sistema, voltados para qualquer aplicação construída para o mesmo.

Por isso, a maneira com que as tarefas são implementadas e a escolha dos mecanismos utilizados provocam também um grande impacto sobre o resultado final obtido pela execução das mesmas.

De maneira bastante simplificada, uma tarefa periódica no linux é constituída por um laço de instruções contendo uma tarefa de tempo real, e a captura do instante de tempo corrente, necessário para o cálculo de intervalo de `sleep` até a próxima ativação.

Enquanto(verdadeiro) `//laço infinito`
dorme até o próximo período;
executa instruções da tarefa de tempo real;
captura o instante corrente;

Em meio a esta pequena seqüência de atividades aparentemente simples, algumas operações importantes estão incluídas. A captura do instante corrente e a chamada que torna a tarefa inativa (em *sleep*) são atividades de baixa precisão no Linux.

O uso de funções do tipo *sleep* na implementação de periodicidade de uma tarefa exige a determinação do intervalo de tempo duranto o qual a tarefa deve permanecer inativa.

As funções do tipo *sleep* implementadas no Linux fazem o uso de um intervalo de tempo como parâmetro. Este, por sua vez, pode ser definido com diferente precisão, conforme a chamada utilizada: `sleep`, `usleep` ou `nanosleep`. Todas elas apresentam funcionamento bastante similar. Embora a chamada `nanosleep` permita especificar um intervalo de tempo com precisão de nanossegundos, o funcionamento desta chamada delimita um tempo mínimo de inativação (tempo atribuído como parâmetro da chamada). Não há garantias quanto ao limite máximo de tempo que a tarefa permanecerá inativa.

Uma vez que este intervalo é variável, deve ser calculado a cada ativação da tarefa. Para que o cálculo seja efetuado, deve ser conhecido os instantes de tempo que determinam o intervalo de inativação. Conforme a figura 4.2, este intervalo (`ts`) tem início no instante de término da instância, e termina no instante da próxima ativação.

O período da tarefa em conjunto com o instante de sua primeira ativação determinam a grade de tempo da mesma. Esta grade de tempo independe de qualquer outra restrição temporal da tarefa, e portanto pode ser deslocada sobre o eixo do tempo conforme o interesse do projetista da aplicação. Este deslocamento é conceituado em [6] como *offset*.

A captura dos instantes de tempo tem papel fundamental no algoritmo da tarefa. Por isso, a precisão do mecanismo de captura do tempo tem influência direta sobre a periodicidade da

mesma. A escolha da fonte de medição é uma etapa importante na implementação da tarefa de tempo real.

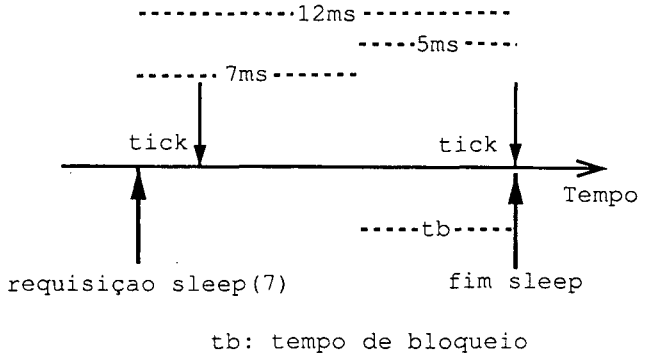


Figura 4.1
Resolução de sleep no Linux

A construção da aplicação de forma que possa ser garantida determinada precisão na medição de tempo e conseqüentemente fornecer base confiável para o estudo desenvolvido, deve apresentar cuidados especiais quanto a escolha da fonte de medição e os valores a serem adotados como parâmetros. O parâmetro utilizado é um valor sobre o qual todos os demais cálculos do programa são efetuados. Ele deve ser lido, preferencialmente de uma fonte de medição de tempo externa ao programa através de consultas, realizadas por chamadas do sistema.

Contadores de hardware caracterizam uma fonte confiável e relativamente precisa de medição de tempo. Trata-se de alguns circuitos de hardware, baseados em osciladores de freqüência fixa e contadores. Os contadores de hardware não são afetados por flutuações de desempenho do sistema, falhas de software ou sobrecarga. Embora o processo de leitura dos mesmos o seja, eles são freqüentemente utilizados como fonte de parâmetros para os programas em questão.

A obtenção do tempo corrente pode ser feito por uma série de funções disponíveis no Linux. A escolha de uma destas funções depende da finalidade, da aplicação, e de características tais como modo de execução (módulos do kernel, aplicação de usuário).

O acesso direto ao contador de *timestamp* é possível através de uma função simples contendo um comando em *assembly* que garante a leitura do valor do contador de *timestamp* da máquina. Para obtenção do valor equivalente ao valor lido convertido em unidades de tempo é necessário o conhecimento, o mais preciso possível da freqüência do processador (o valor geralmente usado é aquele fornecido pelo fabricante indicado em /proc/cpuinfo). A função é ilustrada abaixo:

```
static inline unsigned long long rdtsc() {
  unsigned long long int x;
  __asm__ volatile (".byte 0x0f, 0x31" : "=A" (x));
  return x;
}
```

Foram realizados diversos experimentos com aplicações utilizando a função acima ilustrada na captura dos instantes de tempo. Entretanto, observou-se que a não ser que o valor de freqüência utilizado fosse extremamente próximo do valor real, operação de conversão do valor lido pela função envolve um erro cumulativo que compromete a finalidade da tarefa implementada para o objetivo deste trabalho. A aplicação utilizada em tais experimentos é detalhada em [20].

A periodicidade de uma tarefa no Linux pode também ser implementada a partir de interrupções de dispositivos externos, no caso de tarefas que fazem o uso de comunicação com *drivers* de dispositivo. Este é o mecanismo implementado pela maior parte das variações do Linux para aplicações de tempo real. A leitura dos temporizadores do próprio Linux é feita por chamadas de sistema.

A chamada `gettimeofday` é uma das chamadas de sistema que possibilita a leitura do tempo a partir do modo usuário. As demais chamadas disponíveis foram superadas por `gettimeofday`, porém ainda estão disponíveis no Linux por razões de portabilidade. Esta chamada retorna valores em duas estruturas, `timeval` e `timezone` e é implementada por `sys_gettimeofday`. `sys_gettimeofday`, por sua vez, invoca a função `do_gettimeofday` que executa as seguintes ações:

Copia o conteúdo de `xtime` para o buffer no espaço do usuário especificado pelo parâmetro `tv` da chamada do sistema, atualiza o número de microssegundos, adiciona o número de microssegundos ao tempo correspondente às interrupções do `timer` que não tiveram suas *bottom-halves* executadas e verifica a ocorrência de overflow no campo de microssegundos, ajustando ambos os campos da variável se necessário.

A atualização do número de microssegundos é feito pela invocação da função endereçada pela variável `do_gettimeoffset` que por sua vez aciona a função que lê o registrador TSC usando a instrução *assembly* `rdtsc`. A partir desta leitura, e do conhecimento de outras variáveis, é calculado o número de ciclos executados desde a última interrupção de `timer`. Com tais informações é feita a conversão do tempo em microssegundos. A variável `xtime` é a variável a partir do qual programas de usuário lêem o tempo e a data. Ela armazena o tempo em duas estruturas; `xtime.tv_sec` que guarda o número de segundos passados desde

meia noite de janeiro de 1970 e `xtime.tv_usec`, que armazena o número de microssegundos passados desde o último segundo.

A operação implementada por `gettimeofday` é semelhante à operação a ser realizada caso fosse utilizada no código da aplicação, a leitura direta do contador de *timestamps* (TSC). Porém, tais operações passam a ser executadas no espaço de endereçamento do kernel (usando `gettimeofday`), e a leitura do valor da freqüência de ciclos do processador é lida de uma variável interna ao kernel, cujo valor é calculado no momento de inicialização do sistema.

A periodicidade de uma tarefa exige que esta alterne entre os estados de execução ou inativação por intervalos controlados de tempo. Este requisito pode ser satisfeito através de chamadas de atraso de execução. O padrão POSIX provê chamadas do tipo *sleep* que implementam este atraso. A chamada `nanosleep` do padrão permite especificar intervalos de tempo com precisão de nanossegundos. Porém é importante notar que a granularidade desta chamada não é a mesma da granularidade do temporizador do sistema. Poucos sistemas possuem granularidade suficiente para prover o funcionamento adequado desta chamada. Além deste fator, períodos com interrupções desabilitadas ou com tarefas mais prioritárias ocupando o processador, influenciam sobre o tempo em que a tarefa realmente permanecerá inativa.

Embora o uso de sinais seja uma das várias alternativas válidas na implementação do período de tarefas, no Linux, aplicações com múltiplas *threads* executam exatamente como aplicações comuns (que não fazem o uso de *threads*). Não é possível considerar *threads* independentes quando se trata do uso de sinais na manipulação das mesmas. Além destas considerações, os mecanismos de implementação de sinais está sujeito às mesmas situações das chamadas do tipo *sleep* (interferências, granularidade do sistema).

De qualquer maneira, chamadas a funções de *sleep* são, na maioria dos casos, a melhor opção na implementação de tarefas periódicas de tempo real. Embora sinais também possam ser utilizados, limitações quanto a sua utilização em conjunto com *threads*, aliada a um número maior de fontes de não determinismo envolvidos em sua programação e funcionamento fazem com que este seja um recurso pouco utilizado.

Assim como a grade de tempo descrita por uma tarefa periódica, o timer do Linux implementa uma grade de tempo que determina a ativação de várias atividades do kernel (*flush* de *cache* de disco ou reorganização de áreas livres de memória). Por serem executadas no espaço de endereçamento do kernel, estas atividades, mesmo realizando tarefas de manutenção (sem restrições temporais), têm preferência no escalonador sobre tarefas de tempo real no espaço do usuário. E pelo fato do kernel ser não preemptivo, a existência de uma tarefa de prioridade de tempo real na fila de *runqueue* só é considerada após a finalização das ati-

vidades do kernel no processador. Dessa forma, é caracterizado um atraso inerente ao Linux ao qual a tarefa de tempo real sempre estará exposta.

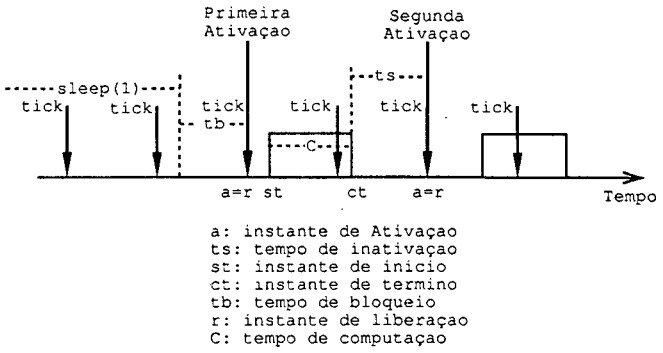


Figura 4.2
Sincronização de grades de tempo

4.2 Aplicações

A construção das aplicações deste trabalho objetivou manter características simples, que pudessem estar contidas no maior número possível de tarefas de tempo real. Por isso, trata-se de aplicações cujo comportamento estará contido em vários tipos de aplicações desta natureza. Ao mesmo tempo, buscou-se fornecer às mesmas, propriedades que permitissem expor estas aplicações a tipos diversos de interferência provenientes da própria aplicação.

As aplicações implementadas são na verdade combinações de tarefas de tempo real e tarefas não tempo real. Cada aplicação foi concebida com um objetivo principal, descrevendo um perfil particular de acordo com o tipo de informação desejada a partir da execução desta tarefa.

Todas as tarefas de tempo real foram implementadas através de *threads* do Linux, mais especificamente, através de *pthreads*(Posix *threads*). *Threads* são definidas como programas em execução ou ainda processos leves e por isso possuem características propícias para implementação de aplicações de tempo real. Por compartilhar espaço de endereçamento e ambiente de execução dentro de um processo, a criação e manutenção de *threads* são realizadas mais fácil e rapidamente permitindo a execução de múltiplas tarefas ao mesmo tempo com o menor custo possível. De fato, *threads* são amplamente utilizadas na área de tempo real.

Em geral, a estrutura de uma aplicação que faz o uso de múltiplas *threads* divide o programa em um processo e as *threads* que o compõe. No processo, estão todos os dados em

comum das *threads* a serem criadas, variáveis globais e atributos que não são explicitamente definidos para cada *thread* são herdados do processo que as origina. Também estão no processo todas as funções que determinam o atributo da *thread* a ser criada, prioridade, política de escalonamento, e demais propriedades que podem ou não ser definidas para a *thread*. O trecho do código da aplicação abaixo ilustra as funções utilizadas na aplicação envolvidas no processo de criação da *thread* de tempo real.

Após a declaração das variáveis necessárias, são definidos os demais dados que fazem parte da criação da *thread*, funções de inicialização relacionam estes dados à *thread* e finalmente uma chamada a `pthread_create` dispara a *thread* com o código descrito pelo terceiro campo de seus argumentos. A *thread* criada utiliza o escalonador SCHED_FIFO com prioridade igual a prioridade máxima desta política menos um. Os demais argumentos definidos para esta *thread* na função `pthread_create` (identificador, número total de instâncias e período) são passados através de seu último argumento.

```
pthread_attr_init(&atributo);
pthread_attr_setinheritsched(&atributo, PTHREAD_EXPLICIT_SCHED);
if(pthread_attr_setschedpolicy(&atributo, SCHED_FIFO)==-1){
      printf("Erro na definicao da politica!\n");
      exit(0);
  };
 mysched.sched_priority = sched_get_priority_max(SCHED_FIFO) -1;
 pthread_attr_setschedparam(&atributo, &mysched);
 retcode = pthread_create(&th[0],  &atributo, (void *)thread, &th_a[0]);
 if (retcode != 0) fprintf(stderr, "create a failed %d\n", retcode);
```

Uma vez que o modelo de escalonamento em uso neste trabalho é o Taxa Monotônica[LiL73], as cinco *threads* de tempo real criadas tem prioridades proporcionais aos seus períodos.

As aplicações são compostas por uma função main, uma função de captura de tempo corrente e ajuste num instante de referência, e o código das *threads* da aplicação. Na função main estão declaradas, além das variáveis utilizadas por todas as *threads*, as funções que permitem a seleção do escalonador das *threads*, a escolha da prioridade de cada uma delas, de acordo com o escalonador escolhido (SCHED_FIFO), além de outras características como período, número de iterações e identificador.

A estrutura geral da *thread* de tempo real é um laço de instruções que contém uma chamada a função de `sleep`, a captura do instante de tempo corrente, um conjunto de instruções

de ponto flutuante e novamente a captura do instante de tempo corrente. Um esboço do código da *thread* é ilustrado abaixo:

```
void * thread(struct thread_arg  *arg)
{
  int i, j, g1;
  struct timespec ts;
  struct timespec tsr;
  struct sched_param this_sched_param;
  struct timeval tv;
  struct timezone tz;
  double sum;
  float avg;

  sum = 0.0;
  j=1;

  sleep(1);
  g1 = gettimeofday(&tv,&tz);
  t0[arg->id]= (tv.tv_sec*DEZNA6) + tv.tv_usec ;

  while(j < arg->ciclos) {
       tp[arg->id][j]=j*arg->period/1000;
       ts.tv_sec = 0;
       sleept[arg->id][j]=( j*arg->period)/1000-agora(t0[arg->id])-18000;
       ts.tv_nsec= sleept[arg->id][j]*1000;
       tsr.tv_sec = 0;
       tsr.tv_nsec = 0;
       nanosleep(&ts,&tsr);

       t1[arg->id][j]=agora(t0[arg->id]);
      /****************************** R ***********************************/

          Código da Tarefa de Tempo Real

      /*********************************************************************/
      t2[arg->id][j]=agora(t0[arg->id]);
```

```
        j++;
    }
        return NULL;
}
```

O propósito da *thread* implementada nas aplicações é simular o comportamento de tarefas periódicas de tempo real além de permitir a captura de instantes de tempo durante sua execução para posterior análise do comportamento temporal da mesma. As informações necessárias e as etapas seguidas para a composição desta tarefa, bem como os cuidados tomados e o conhecimento adquirido durante a concepção destas atividades servem como base para a construção de qualquer tarefa de tempo real no Linux.

Como se trata de uma aplicação hipotética, o corpo da tarefa de tempo real construída para os experimentos não apresenta nenhum propósito específico, mas deve simbolizar uma seqüência de operações que farão uso de tempo do processador. Por isso, o corpo da tarefa em questão é composto por um laço de operações de ponto flutuante.

O corpo da *thread* que implementa a tarefa de tempo real foi mantida o mais simples possível. Um dos pontos de projeto destas *threads*, foi manter um código que permitisse o mínimo de variação de comportamento, e utilizasse instruções simples que provocasse o mínimo de incerteza. O valor do instante corrente é capturado no início e ao término da execução do laço que representa a tarefa de tempo real. Este intervalo de tempo caracteriza o tempo de computação da tarefa somado a possíveis interferências sofridas pela mesma durante sua execução e é referenciado por IF no decorrer deste trabalho.

Testes realizados com as primeiras aplicações implementadas forneciam valores de *jitter* de chegada bastante variados. Cada sessão parecia adotar um valor de *jitter* de chegada diferente, independente do valor da sessão anterior. A aleatoriedade dos valores apresentados levantou questões a respeito da sincronização das grades de tempo descritas pela tarefa de tempo real e da grade de tempo descrita pelo *timer* do Linux.

Observando o comportamento da chamada do tipo *sleep* e os resultados obtidos em estudos preliminares com outras aplicações anteriormente implementadas, verificou-se a possibilidade de empregar esta chamada para realizar a sincronização entre as grades de tempo (da tarefa e do *timer* do Linux).

Assim, logo após a inicialização das variáveis usadas pela *thread*, uma chamada *sleep* com parâmetro 1 segundo é invocada. A chamada faz com que a tarefa permaneça inativa por aproximadamente 1 segundo, e volta a tornar ativa após o término do tempo de `sleep`. O instante de retorno da tarefa à fila de prontos é estabelecida de acordo com a grade de

interrupções do `timer` (como ocorre com qualquer chamada tipo *sleep*), e a probabilidade desta ocorrer o mais próximo possível do início de um novo período de interrupção é maior (figura 4.2).

A *thread* implementada para as aplicações deste trabalho faz uso da chamada `nanosleep`. A função `nanosleep` usa como parâmetro, um valor de tempo em nanossegundos. Ainda assim, a precisão nesta magnitude está longe de ser alcançada no Linux. Ao fornecer a esta chamada um valor x, ela garante a inativação da tarefa por pelo menos x nanossegundos, não fornecendo qualquer garantia quanto ao valor máximo que pode ser alcançado por esta chamada, conforme descrito anteriormente.

O período da tarefa da aplicação construída é implementado na *thread* através da combinação das chamadas de `gettimeofday` e da chamada `nanosleep`, que usa como parâmetro o intervalo de tempo entre o instante de fim de execução da instância da tarefa e o tempo de chegada da próxima instância, que coincide com o início do próximo período. Basicamente, a chamada `nanosleep` é executada usando como parâmetro o instante em que a tarefa deve acordar (início do próximo período) menos o instante corrente (lido no momento em que esta termina sua execução).

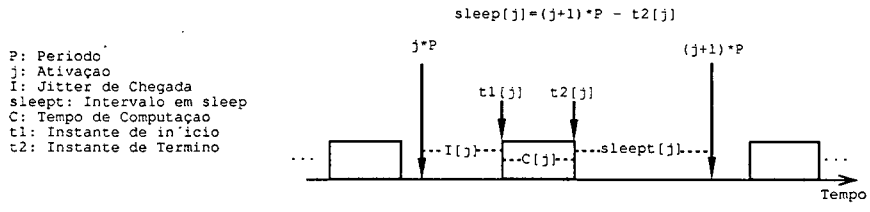


Figura 4.3
Cálculo de intervalo de sleep

O estudo sobre o comportamento das funções do tipo *sleep* mostrou também que estas chamadas estão sujeitas a um atraso constante. Mesmo efetuando a sincronização das grades da tarefa e do sistema, o tempo fornecido como parâmetro da chamada e o tempo durante a qual a tarefa realmente permanece inativa apresentam uma diferença constante durante todas as ativações da tarefa. Uma vez encontrado um valor compensatório, pode-se aproximar o tempo de *jitter* de chegada das ativações de zero. O valor compensatório é encontrado através de métodos empíricos, e varia de acordo com o ambiente de execução da tarefa. A descrição dos experimentos realizados na busca do valor compensatório são detalhadas em [20].

De acordo com o resultado obtido a partir destes experimentos, foi encontrado para a aplicação implementada, o valor compensatório de 18ms. Este valor foi encontrado após várias sessões de teste com as aplicações implementadas.

Após a modificação do cálculo do parâmetro da chamada `nanosleep`, o *jitter* de chegada apresentado pelas aplicações tornaram-se mais próximos de 0. Porém, não é possível garantir o ajuste perfeito deste intervalo, o que traz conseqüências tais como intervalos de tempo de valor negativo. O valor negativo apresentado indica a quantidade de tempo antes do qual a ocorrência esperada foi executada. Um valor -x medido para o tempo de chegada de uma instância indica a chegada da mesma x microssegundos antes do início de seu período. Conseqüentemente, caso o início e o fim da tarefa ocorram antes do início do período da mesma, o tempo de resposta desta tarefa também é dado por valores negativos. O ajuste de 18ms foi o mais próximo possível que se pode alcançar da obtenção de tempos de chegada próximos de zero. Na tabela 4.2 estão listados os tempos de início oficial e relativos obtidos pela execução das tarefas com o valor compensatório.

Grades de tempo ajustadas (tempos em microssegundos)		
Instante de Ativação	Instante de Início	Instante de Início Relativo
200000	199935	-65
400000	399929	-71
600000	599931	-69
800000	799931	-69
1000000	1000001	1
1200000	1199927	-73
1400000	1399925	-75
1600000	1599926	-74
1800000	1799927	-73
2000000	1999924	-76
2200000	2199925	-75
2400000	2399925	-75
2600000	2599927	-73
2800000	2799943	-57
3000000	2999936	-64
3200000	3199927	-73
3400000	3399924	-76
3600000	3599927	-73
3800000	3799927	-73
4000000	3999925	-75

Tabela 4.1: Listagem parcial de R com ajuste de grade de tempo.

Porém, ao verificar o comportamento das aplicações diante de cenários mais complexos, percebeu-se que a interferência sofrida pelas tarefas de tempo real não permite o ajuste do tempo de *sleep* com resultados muito satisfatórios. O valor compensatório varia de acordo com o ambiente de execução da tarefa. Mesmo assim, o ajuste de *sleep* foi mantido a fim de estabelecer condição padrão para todas as aplicações utilizadas nos experimentos. Sob estas condições, adota-se que os tempos medidos; tempo de resposta máximo e tempo de resposta mínimo, indicados em tabelas subseqüentes fornecem valores que servem como

pontos de referência. Como mencionado anteriormente no capítulo, os tempos de chegada, determinados pela grade de tempo da tarefa são determinados na primeira ativação da tarefa, e podem ser deslocadas sobre o eixo do tempo de acordo com o interesse do projetista da aplicação. Portanto, caso exista interesse em que os valores coletados pela aplicação sejam positivos, basta efetuar o deslocamento da grade de tempo da tarefa (*offset*).

É importante salientar que embora o valor compensatório varie de acordo com o ambiente de execução, não existe qualquer espécie de erro cumulativo. Tanto números gerados pelos programas quanto gráficos e experimentos empíricos mostram que o valor compensatório não acarreta em qualquer alteração cumulativa dos tempos de resposta da aplicação.

Threads comuns tem por objetivo representar a parte não tempo real da aplicação. Efetuando tarefas simples ou tarefas de uso intenso de dispositivos, elas permitem verificar o quanto as tarefas de tempo real são afetadas por tarefas comuns, mesmo tendo estas, prioridade menor.

Quatro *threads* não tempo real foram implementadas. As *threads* não tempo real de teclado e vídeo são tarefas triviais, descritas em poucas linhas de código. Estas tarefas estão geralmente presentes em aplicações simples que exigem dados de entrada do operador da aplicação ou a exibição de dados gerados.

A *thread* de teclado simplesmente utiliza uma função `getc` para capturar a entrada através do deste dispositivo, uma vez acionada, esta *thread* permanece em execução até que uma tecla seja pressionada ou até que todas as *threads* de tempo real terminem sua execução. A *thread* de impressão na tela imprime, dentro de um laço de repetição, um caracter qualquer no vídeo, assim como a *thread* de tecla, ela permanece em execução até que as *threads* de tempo real terminem.

As *threads* de rede e disco são mais complexas por utilizarem recursos que exigem código mais sofisticado. Ainda assim, implementam funções básicas sobre estes dispositivos. A tarefa de acesso a disco é uma tarefa periódica de período igual a 100ms. Esta tarefa simplesmente abre e lê uma série de arquivos de tamanho próximo ao tamanho da memória principal da máquina em que executa.

A *thread* de requisição de pacotes de rede implementa um laço de comandos que requisita arquivos html do servidor local. Esta não é uma *thread* periódica, e não existe qualquer ação sobre o pacote após o seu recebimento. Assim como a *thread* de acesso a disco, a *thread* de acesso a rede não é escalonada pelo escalonador SCHED_RR ou SCHED_FIFO. O escalonador usado para esta tarefa é o escalonador padrão do Linux (SCHED_OTHER).

A aplicação mais simples do conjunto, implementa uma única *thread* de tempo real e é chamada de TR1. A tarefa de TR1 possui período igual a 50ms e determina uma sessão

de experimento com um total de 3000 instâncias. Esta aplicação básica permite observar o comportamento do sistema quando dedicado somente a execução de uma única e de alta prioridade, tarefa de tempo real.

A segunda aplicação implementada é a aplicação TR7. Esta aplicação implementa cinco *threads* de tempo real e duas *threads* não tempo real. As cinco *threads* tempo real utilizam, cada uma, um valor de período diferente, e prioridade definida de acordo com escalonamento Taxa Monotônica apresentada no capítulo 2. As duas *threads* não tempo real implementadas nesta aplicação são as duas *threads* mais simples, de teclado e vídeo. TR7 permite verificar o comportamento de tarefas de tempo real na presença de outras tarefas no sistema. Uma vez verificado que a interferência das duas tarefas não tempo real sobre elas é mínima, a maior interferência provocada sobre a tarefa mais prioritária será provocada pelas demais tarefas da própria aplicação TR7.

A maior aplicação construída para os experimentos é a aplicação TR9. A aplicação TR9 é composta por cinco *threads* de tempo real implementadas da mesma maneira que em TR7 e quatro *threads* não tempo real; *thread* de teclado, *thread* de vídeo, *thread* de acesso a disco e *thread* de acesso a rede. Ela permite verificar o impacto provocado pela presença de tarefas no sistema que fazem o uso intensivo de dispositivos como rede e disco e que ocasionam por sua vez a execução de funções dentro do kernel. Desta maneira, funções acionadas por tarefas comuns passam a interferir sobre tarefas de tempo real através da ativação de funções do kernel.

4.3 Cenários

Os cenários compostos para condução dos experimentos visam fornecer informações sobre o comportamento temporal das *threads* de tempo real do Linux quando submetidas a diferentes condições de execução. Dada a amplitude de aplicabilidade de tarefas de tempo real, buscou-se construir cenários que ao mesmo tempo, constituíssem ambientes comumente encontrados e que pudessem gerar informações válidas para a maior variedade possível de aplicações.

Basicamente, três tipos de cenários são utilizados: cenários de execução de aplicação com uma única *thread* de tempo real; cenários com várias *threads* de tempo real e *threads* não tempo real compondo uma aplicação; e cenários compostos pelos cenários anteriores na presença de carga adicional no sistema. No total, foram conduzidos experimentos em nove cenários distintos. Não foi realizado testes de escalonabilidade sobre o conjunto de tarefas que compõe cada cenário. Embora os tempos de computação das tarefas sejam pouco

variáveis, informações quanto a escalonabilidade do conjunto de tarefas são obtidos apenas a partir da execução das mesmas, cujo resultado é apresentado no capítulo 5.

Os dois primeiros cenários são os cenários mais simples, que executam a aplicação TR1 sem qualquer carga adicional no sistema. O primeiro cenário, texto-TR1, é o cenário em que a única *thread* de tempo real executa em modo texto. Neste cenário, além da própria aplicação, só estão presentes no sistema os processos pertencentes ao próprio Linux. O segundo cenário, gui-TR1, é análogo ao cenário texto-TR1, porém, a execução da aplicação é feita através de uma janela na interface gráfica de usuário. Uma pequena interferência é esperada deste cenário em comparação com o cenário texto-TR9.

A mesma classificação é dada às demais aplicações, TR7 e TR9. Ambas as tarefas são executadas em cenários compostos pela aplicação em modo texto e pela aplicação sobre interface gráfica de usuário. Estes cenários serão referenciados por texto-TR7, gui-TR7, texto-TR9 e gui-TR9. Trata-se de cenários sem nenhuma carga adicional. As cinco tarefas de tempo real que compõem estas aplicações utilizam o escalonador SCHED_FIFO com prioridades diferentes, sendo a maior prioridade da primeira tarefa e a menor prioridade de conjunto, da quinta tarefa. O trecho da aplicação abaixo ilustra a distribuição de períodos, identificadores e número total de ativações de cada uma das tarefas. Estas estruturas definidas no processo da aplicação são passadas como argumentos na criação de cada *thread* de tempo real.

```
th_a[0].id=0;
th_a[0].period=50000000;
th_a[0].ciclos=3000;

th_a[1].id=1;
th_a[1].period=150000000;
th_a[1].ciclos=1000;

th_a[2].id=2;
th_a[2].period=200000000;
th_a[2].ciclos=750;

th_a[3].id=3;
th_a[3].period=250000000;
th_a[3].ciclos=600;

th_a[4].id=4;
```

```
th_a[4].period=300000000;
th_a[4].ciclos=500;
```

Os cenários com interface gráfica de usuário descritos acima são utilizados na composição dos cenários com carga. Estes cenários executam, além da interface de usuário e da aplicação de tempo real, programas comuns de usuário caracterizando carga adicional. Os programas de carga adicional fazem constantes requisições de pacotes de rede e uso intensivo de processador. A tarefa da carga adicional de uso de dispositivo de rede é idêntico ao implementado na aplicação TR9, executa um laço infinito de instruções de requisição de pacotes de rede de um servidor local. A *thread* que simula uso intenso de processador é concebida por um laço de compilação de um kernel e seus módulos.

Componentes	**Aplicações**		
	TR1	TR7	TR9
Modo Texto	texto-TR1	texto-TR7	texto-TR9
Interface Gráfica de Usuário	gui-TR1	gui-TR7	gui-TR9
Interface Gráfica de Usuário com Carga Adicional	gui-TR1carga	gui-TR7carga	gui-TR9carga

Tabela 4.2: Cenários

4.4 Conclusão

Buscando fornecer resultados válidos para os experimentos conduzidos neste trabalho, além de várias considerações a respeito do sistema operacional, uma série de cuidados em relação a implementação das tarefas foram tomados. Foi observado, durante a construção de aplicações e cenários de experimentos que o sistema operacional impõe restrições ao desempenho das tarefas de tempo real que não permitem a verificação de todos os elementos existentes no modelo de tarefas periódicas.

Embora o modelo de tarefas periódicas mostre claramente todos os elementos necessários a descrição do comportamento temporal de uma tarefa periódica, não existem meios para que a implementação desta classe de tarefas seja feita de modo determinista. A riqueza de informações utilizadas no modelo só pode ser adquirida quando todos os elementos envolvidos na execução da tarefa podem ser analisado como uma caixa de vidro, quando é possível saber exatamente sobre o comportamento temporal de todos os elementos envolvidos com a execução da tarefa.

No decorrer da implementação das aplicações utilizadas, e na construção e seleção dos cenários dos experimentos, muitos outros experimentos foram realizados. A escolha de cada elemento que compõe as aplicações e cenários foram baseadas em informações e observações adquiridas durante o tempo de construção das mesmas. Estes dados nem sempre foram aproveitados para os experimentos ilustrados posteriormente, mas permitiram direta ou indiretamente a melhora da qualidade dos mesmos.

Mesmo fazendo uso dos mecanismos implementados no próprio Linux para obter a melhoria do comportamento do mesmo para tarefas de tempo real, as limitações deste sistema, voltado para aplicações gerais não permite a programação e execução de tarefas com restrições temporais rígidas. Os mecanismos existentes no Linux que são voltados para aplicações de tempo real representam apenas um esforço de melhorar o desempenho do sistema para tarefas desta natureza. O efeito das limitações do Linux não pode ser significativamente amenizado pela implementação de mecanismos simples. Além disso, a maneira com que a *thread* de tempo real deve ser implementada para apresentar o comportamento próximo do desejado insere indeterminismo em seu comportamento.

Algumas das limitações do Linux não se restringem a funções ou dispositivos, trata-se de características inerentes ao sistema cujos efeitos refletem sobre tudo o que executa no mesmo. Embora tais limitações imponham características às aplicações que as tornam impróprias para sistemas de tempo real hard, elas não impedem seu bom desempenho para uma grande quantidade de sistemas de tempo real soft.

A resolução de interrupções de `timer` do Linux é em especial um fator de grande influência no desempenho do mesmo para aplicações de tempo real. A precisão deste temporizador possui impacto sobre todas as chamadas utilizadas pelas aplicações construídas sobre ele. Ainda assim, este não é o único contratempo encontrado durante a implementação de tarefas de tempo real no Linux. O fato do kernel ser não preemptivo provoca interferências relativamente longas sobre as tarefas de tempo real.

Ainda assim, a partir do estudo dos mecanismos utilizados nas aplicações é possível utilizar os recursos do próprio sistema para melhorar os resultados que se deseja, mesmo que estes não sejam ideais. A combinação deste estudo juntamente com experimentos baseados nos mesmos fornecem bons indicativos a respeito do comportamento do Linux e como conseguir um melhor desempenho deste.

Este capítulo apresentou algumas observações e informações coletadas durante a construção de esqueletos de aplicações de tempo real. Foram apresentados ainda as aplicações a serem utilizadas nos experimentos do próximo capítulo bem como os cenários em que estes executam.

Capítulo 5

Resultados das Experiências

O Linux é uma plataforma que apresenta um bom desempenho para aplicações de propósitos gerais. Suas propriedades, entretanto, o tornaram um sistema bastante atraente para projetistas de aplicações de tempo real. Tais aplicações, apresentam requisitos muitas vezes conflitantes com alguns aspectos do Linux. Mas ainda assim, esta plataforma apresenta comportamento satisfatório para algumas aplicações com restrições temporais brandas.

Neste capítulo são ilustrados os resultados obtidos pela execução das aplicações implementadas nos cenários selecionados. Os dados apresentados foram obtidos em um PC com processador AMD K6-II 500MHz com 128Mbytes de memória, com Linux 2.4.0 test10.

Os dados capturados em cada sessão de experimento são fornecidos pelas tarefas de tempo real sob a forma de seqüências de valores de tempo. São fornecidos em cada sessão, o instante de início, o instante de término e o tempo de resposta da tarefa. Cada sessão de experimento é caracterizada por uma execução da aplicação de tempo real. Os dados que baseiam as afirmações deste capítulo foram extraídos de algumas centenas de sessões de experimento.

A apresentação dos resultados é feita na forma de tabelas e gráficos. As tabelas ilustram os valores máximo e mínimo, a variação e o desvio padrão dos tempos de resposta. As unidades utilizadas, tanto nas tabelas quanto nos gráficos são: tempo em microssegundos e instâncias de tarefas de tempo real.

Os gráficos foram construídos através da ferramenta MATLAB, a partir das seqüências de valores resultantes de cada sessão. Dois tipos de gráficos foram construídos para cada tarefa: gráfico de seqüência e histograma. Os gráficos de seqüência ilustram a seqüência do tempo de resposta de cada instância da tarefa no tempo. O eixo X do gráfico contém a seqüência de

instâncias da tarefa. O eixo Y deste gráfico contém valores de tempo de resposta. Portanto, cada ponto do gráfico corresponde ao tempo de resposta y da instância x.

Gráficos de seqüência permitem verificar a existência ou não de um padrão no comportamento da tarefa. Alguns cenários proporcionam repetições periódicas de valores de tempo de resposta (aproximados), gerando gráficos uniformes quanto a curva descrita pelos valores de R. Outros porém, ocasionam a geração de gráficos que apresentam picos de valores de tempo de resposta bem maiores que os valores encontrados na maior parte do gráfico. E outros ainda apresentam curvas sem qualquer regularidade. O formato da curva descrita pelo gráfico de seqüência varia conforme a carga apresentada no sistema durante as medições. Sessões de experimento de um mesmo cenário apresentam, em geral, curvas semelhantes em seus gráficos de seqüência, permitindo observações gerais, baseadas em gráficos de seqüência, sobre experimentos em um mesmo cenário.

Os gráficos de seqüência não serão integralmente ilustrados. Somente um trecho de cada um destes gráficos foi selecionado, correspondendo a tempos de resposta de uma seqüência de 400 ativações.

O gráfico de histograma define a distribuição dos tempos de resposta em grupos. Os limites que delimitam cada grupo são estabelecidos por um valor (*beans*) adotado durante a construção do gráfico. Este valor determina o número de grupos entre os quais os tempos de resposta gerados são distribuídos. Nos histogramas deste capítulo, o valor adotado durante a construção foi 200 *beans*. Sendo assim, os tempos de resposta gerados são distribuídos entre 200 grupos de valores no histograma. Cada barra do gráfico de histograma indica a freqüência y de ocorrências de instâncias com tempo de resposta associado ao *bean* x. Uma vez que os valores de R variam de acordo com a tarefa da aplicação, o *bean* x pode indicar um único valor de tempo de resposta, ou um conjunto de valores de tempo de resposta.

Um gráfico de histograma usando 90% das instâncias de cada tarefa é construído com a intenção de permitir a melhor visualização de distribuição de instâncias de acordo com seus tempos de resposta. As instâncias escolhidas para a construção deste histograma parcial da tarefa são aquelas cujo tempo de resposta se encontra mais distante dos valores limites apresentados na sessão de experimento. Portanto, após a classificação das instâncias de acordo com o valor de R, 5% das mesmas que apresentam valor mais alto e 5% das que apresentam valor mais baixo são excluídas do conjunto que compõe o histograma parcial.

Os valores correspondentes à 90% das instâncias restantes são também indicados pelas tabelas.

Esse capítulo é organizado em 5 seções. As seções 5.1, 5.2 e 5.3 descrevem, respectivamente, a execução das aplicações de tempo real TR1 (1 *thread*), TR7 (7 *threads*) e TR9 (9

threads) em cenários sem carga adicional. Em cada uma destas seções é detalhado o resultado do experimento com a aplicação, a partir da descrição do comportamento da execução em cenário utilizando modo texto e cenário utilizando interface gráfica.

A seção 5.4 descreve os resultados obtidos mediante a execução das três aplicações nos cenários com carga adicional. Os eventuais desvios de comportamento, diferenças apresentadas no comportamento das aplicações diante dos mesmos cenários porém submetidos a outro processador serão descritos na sessão 5.5, a última seção deste capítulo apresenta uma breve conclusão do capítulo apontando as principais informações obtidas pelos experimentos realizados.

5.1 Aplicação TR1: cenários texto-TR1 e gui-TR1

Os experimentos descritos nesta seção caracterizam as aplicações mais simples de tempo real. O cenário texto-TR1 é composto pela aplicação TR1 de única *thread* de tempo real em execução em modo texto no sistema. O cenário gui-TR1 é composto por TR1 executando a partir de um terminal na interface gráfica de usuário. Em ambos os cenários não existe carga adicional no sistema, isto é, não existem outras aplicações em execução.

Os dados obtidos a partir do experimento conduzido no cenário texto-TR1 foram usados para a construção dos gráficos 5.1, 5.2 e 5.3. O gráfico 5.1, é um gráfico de seqüência parcial de TR1, ilustrando as instâncias 500 a 900 do total de 3000 instâncias executadas pela tarefa.

Pelo gráfico de seqüência desta tarefa, é observado que o comportamento da mesma é uniforme na maior parte do tempo. Os tempos de resposta das instâncias variam poucos microssegundos durante a maior parte da sessão de experimento.

Em algumas sessões de experimento foi observada a ocorrência de uma ativação com tempo de resposta bem maior que o tempo de resposta típico da tarefa. Nos cenários com a aplicação TR1 sem carga adicional, esta ocorrência, que caracteriza um tempo de resposta máximo da sessão chegou a apresentar um valor de aproximadamente 10ms.

Todas as aplicações descritas neste capítulo estiveram sujeitas a esta ocorrência. A instância com o tempo de resposta máximo (muito maior que todas as demais ativações) ocorre freqüentemente, porém não ocorre em todas as sessões.

A tabela 5.1 apresenta os tempo de resposta mínimo e máximo apresentados pela tarefa de TR1, bem como a variação dos mesmos e o desvio padrão. Os mesmos dados, referentes a execução de 90% das ativações da tarefa são também indicados na tabela.

O gráfico 5.2, histograma de TR1, ilustra a distribuição dos tempos de resposta desta tarefa. A concentração da curva no lado esquerdo do gráfico indica que o valor de R de maior ocorrência está mais próximo do limite mínimo de tempo de resposta apresentado pela tarefa. A extensão do gráfico até o valor de 30us, indica a existência de valores de R próximos a este valor limite no experimento. Devido a proporção de ocorrências com estes valores, a visualização da cauda da curva do gráfico é dificultada porém, o eixo x do gráfico de histograma sempre considera a distribuição de todos os valores de tempo apresentados durante todo o experimento.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-48	21	69	2.47	-48	-44	4	0.8
T0	-85	-8	77	8.33	-85	-55	30	5.25

Tabela 5.1: texto-TR1 e gui-TR1 (Valores em microssegundos)

O gráfico 5.3 ilustra o histograma parcial da tarefa T0, construído com 2700 instâncias desta tarefa (90% do total de instâncias). Os valores de tempo que delimitam a faixa de R do experimento estão indicados na tabela 5.1, juntamente com os demais valores referentes a esta sessão de experimento.

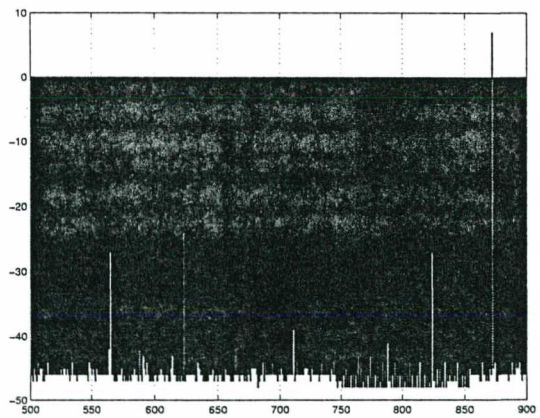


Figura 5.1
Gráfico de R - texto-TR1

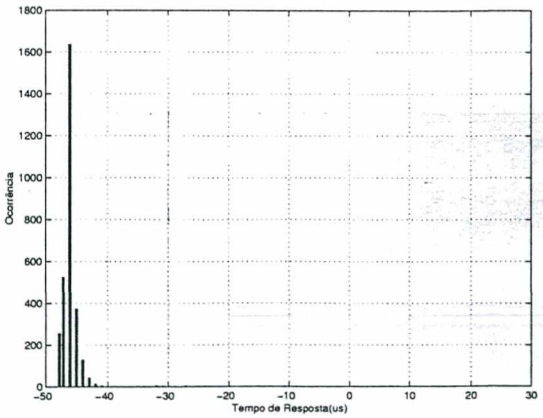


Figura 5.2
Histograma de R - texto-TR1

O tempo capturado entre o instante de início e a conclusão da execução da tarefa varia entre 5us e 58us. Entretanto, os tempos maiores que 8us ocorrem somente em três das instâncias executadas, que correspondem às três instâncias com maior valor de tempo de resposta. Embora os tempos de IF das tarefas apresentem uma determinada variação, freqüentemente, os valores máximos deste intervalo, ocorrem com pouca freqüência. Em cenários cujos

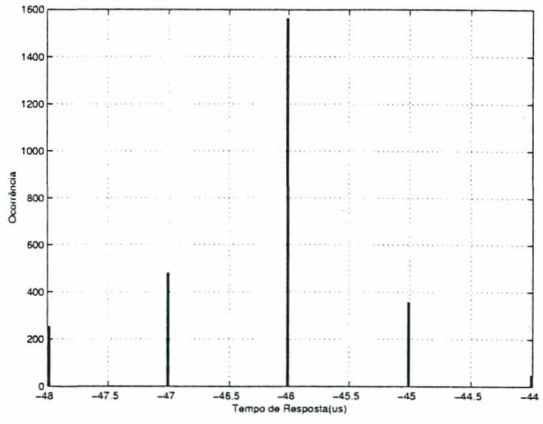


Figura 5.3
Histograma parcial de R - texto-TR1

resultados são apresentados em microssegundos, estas ocorrências caracterizam menos de 10% do total de ativações.

O cenário gui-TR1 é equivalente ao cenário anteriormente descrito, porém inclui a execução de interface gráfica de usuário. Os resultados obtidos neste cenário são ilustrados nos gráficos 5.4, 5.5 e 5.6. Os resultados obtidos neste cenário apresentam um aumento de tempos de resposta em comparação com o cenário anterior (texto-TR1), caracterizando interferências provocadas por processos responsáveis pela interface gráfica de usuário sobre as tarefas de tempo real.

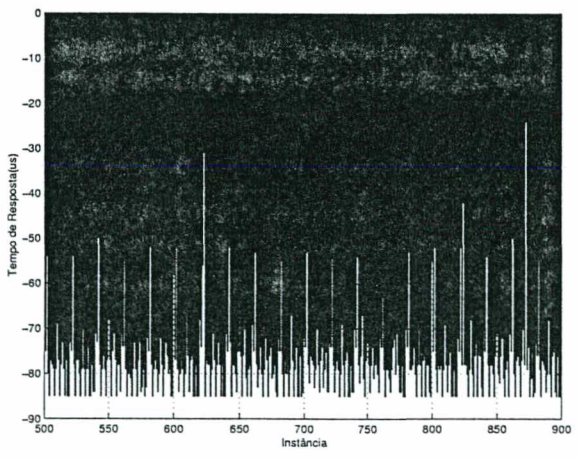


Figura 5.4
Gráfico de R - gui-TR1

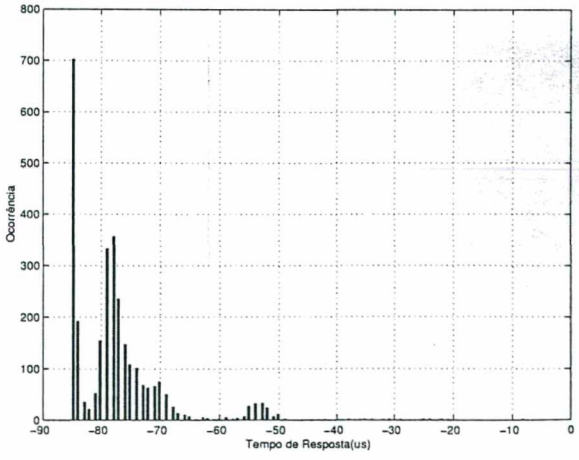


Figura 5.5
Histograma de R - gui-TR1

Ao comparar os os gráficos de seqüência dos dois cenários; texto-TR1 e gui-TR1 verifica-

se a ocorrência de perturbações de tempos de resposta em pontos semelhantes do gráfico. A cada período de aproximadamente 10 segundos, a instância da tarefa apresenta tempo de resposta de dezenas de microssegundos maior que as demais. A causa da perturbação parece ser a mesma em ambos os cenários. Apesar do aumento da variação de tempo de resposta da tarefa em cenário com interface gráfica, não foram observadas ocorrências de maiores perturbações além daquelas já apresentadas pelo gráfico do cenário em modo texto. Os gráficos 5.4 e 5.4 ilustram parte do gráfico de seqüência dos dois cenários com TR1.

A tabela 5.1 indica os valores de tempo de resposta mínimo e máximo para esta tarefa, bem como a variação destes tempos e o valor de desvio padrão calculado para esta variação. Os mesmos valores são indicados também quando considerado somente 90% do total de instâncias da tarefa.

Embora a variação de valores de R para a tarefa em gui-TR1 não seja muito distante da faixa de valores de texto-TR1, a distribuição dos mesmos resulta em gráficos de histogramas com curvas diferentes.

O histograma de texto-TR1(gráfico 5.2) indica uma única concentração de ocorrências de valores de R. O histograma de gui-TR1(figura 5.5) entretanto, apresenta ocorrências mais dispersas no domínio do tempo de resposta e a concentração de ocorrências, embora em menor quantidade, em torno de valores diferentes de tempo (não concentradas num único pico).

A partir da exclusão de 10% dos valores mais próximos do tempo de resposta máximo e mínimo da tarefa, obtém-se o histograma da figura 5.6.

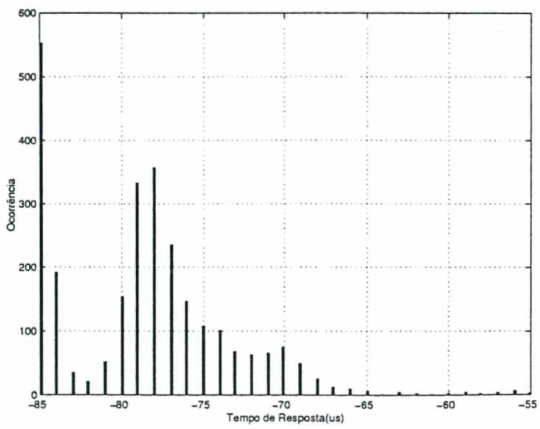


Figura 5.6
Histograma parcial de R - gui-TR1

Os tempos entre o início e o fim da execução (IF) das instâncias apresentados neste ce-

nário são muito próximos daqueles apresentados pelo cenário anterior. Apesar de existirem tempos de IF maiores que 7us, estes totalizam apenas três ocorrências, que são correspondentes à instâncias com tempo de resposta mais altos da sessão de experimento.

A presença de interface gráfica no ambiente de execução desta aplicação provoca um pequeno aumento na variação de R da tarefa, o que permite verificar que realmente existe a interferência de tarefas comuns sobre a execução de tarefas de tempo real. Considerando que os tempos de resposta em texto-TR1 e gui-TR1 são expressos em microssegundos, a interferência provocada pela interface gráfica se mostra presente porém não pode ser considerada significativa no contexto do Linux dada resolução adotada no sistema.

5.2 Aplicação TR7: cenário texto-TR7 e gui-TR7

Esta seção descreve os resultados obtidos pela execução da aplicação TR7 executando em modo texto e com interface gráfica de usuário no sistema. Estas condições de execução caracterizam os cenários texto-TR7 e gui-TR7 respectivamente.

Embora a aplicação TR7 seja composta por sete *threads*, duas destas *threads* são *threads* não tempo real, e portanto não geram informações de interesse para o objetivo destes experimentos. Resultados fornecidos por estas *threads*, estarão presentes nos tempos obtidos pelas *threads* de tempo real sob a forma de interferências nos valores das mesmos.

Os dados relacionados à sessão de experimento cujo gráfico 5.7 é baseado, são ilustrados na tabela 5.2.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-106	-24	82	7.23	-102	-82	20	5.02
T1	-139	-38	131	10.34	-131	-107	23	5.76
T2	-170	-50	120	17.96	-162	-111	51	15.10
T3	-200	-72	128	22.23	-193	-118	75	17.98
T4	-202	-68	134	24.52	-189	-110	79	20.29

Tabela 5.2: texto-TR7 (Valores em microssegundos)

A primeira *thread* da aplicação TR7, é criada com identificador 0. A *thread* T0 é a *thread* de mais alta prioridade do conjunto que compõe TR7. Seu período é de 50ms, e o número total de instâncias executadas é 3000. T0 apresenta os mesmos atributos da tarefa de tempo real implementada por TR1. Os graficos 5.7, 5.8 e 5.9 correspondem aos tempos de resposta obtidos pela execução desta tarefa.

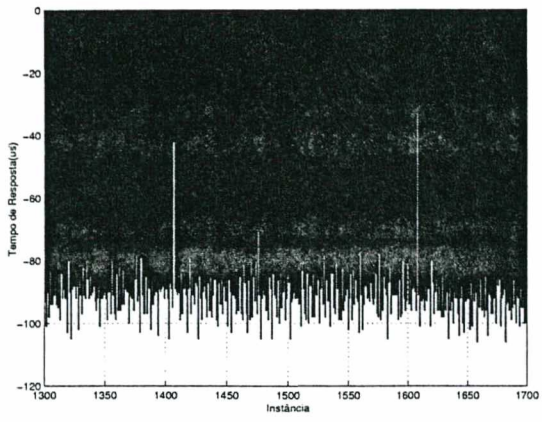


Figura 5.7
Gráfico de R - texto-TR7 T0

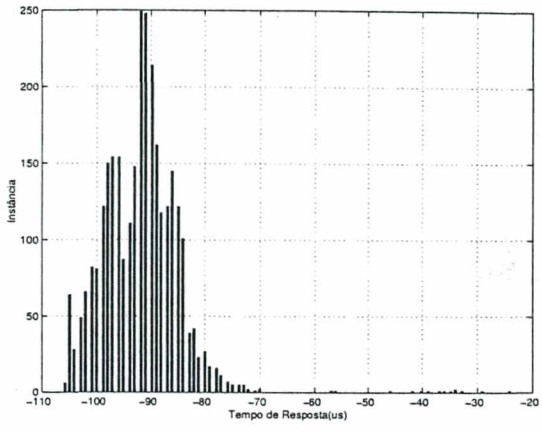


Figura 5.8
Histograma de R - texto-TR7 T0

Comparando T0 com a tarefa de TR1, é percebido um pequeno aumento na variação de valores assumidos pelos tempos de resposta das ativações. Uma vez que a diferença entre os cenários texto-TR1 e texto-TR7 reside apenas nas demais tarefas da própria aplicação TR7, conclui-se que o aumento apresentado em T0 é decorrente de interferências provocadas por tarefas de TR7.

Como observado no histograma 5.8, a distribuição de tempos de resposta de T0 descreve uma curva que lembra a curva de distribuição normal, com valores intermediários de R ocorrendo com maior freqüência que os valores extremos.

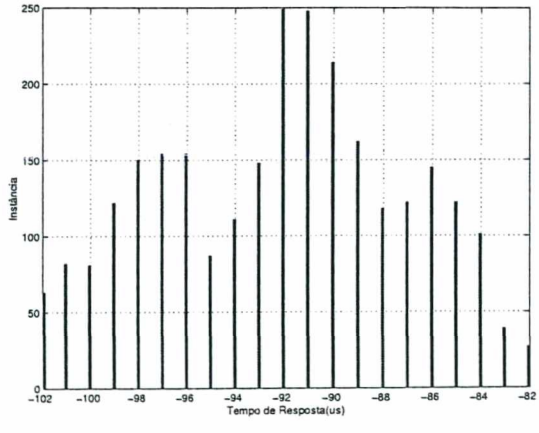


Figura 5.9
Histograma parcial de R - texto-TR7 T0

Ao excluir do conjunto de instâncias que compõem o gráfico de histograma, 10% das

ocorrências com tempo de resposta próximos aos valores limites, obtém-se o histograma da figura 5.9. Neste histograma parcial de T0, que considera 2700 instâncias da tarefa, verifica-se que a maior parte da cauda do histograma de 5.8 desaparece. A tabela 5.2 indica os números relacionados ao experimento deste cenário.

A segunda tarefa de maior prioridade de TR7, com identificador 1, possui período igual a 150ms. Assim como T0, T1 também apresenta um padrão quanto aos tempos de resposta que descrevem a curva do gráfico de seqüência 5.10 desta tarefa. A primeira vista, o gráfico de seqüência de T1 é bastante parecido com o gráfico de seqüência de T0.

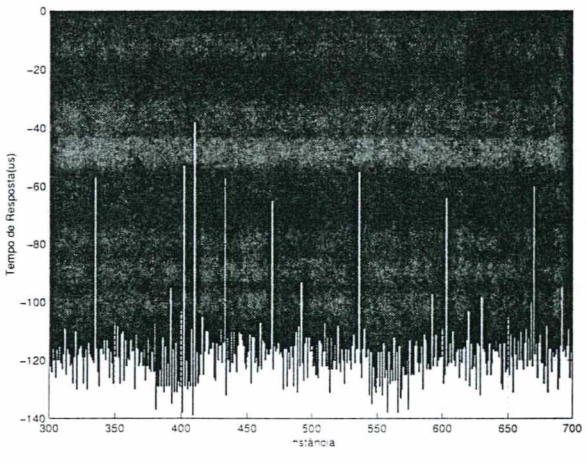


Figura 5.10
Gráfico de R - texto-TR7 T1

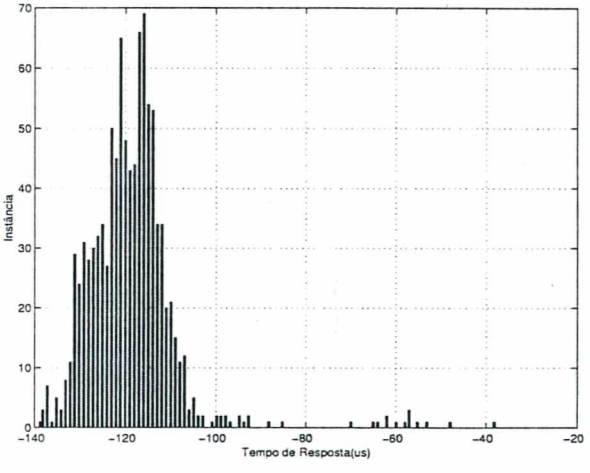


Figura 5.11
Histograma de R -texto-TR7 T1

O comportamento da tarefa T1 não difere em muito do comportamento de T0 neste cenário. A tarefa fornece tempos de resposta em microssegundos, conforme indicado na tabela 5.2, com valores pouco maiores que aqueles apresentados por T0.

O histograma de T1(gráfico 5.11) apresenta a concentração de ocorrência dentro de um espaço de tempo maior que no histograma de T0(gráfico 5.8), apesar da curva de distribuição ser semelhante.

O histograma baseado no conjunto de 90% das instâncias é ilustrado pelo gráfico 5.12, em que é possível ter a melhor percepção sobre a distribuição de ocorrências no domínio de valores de tempo de resposta de T1. Demais dados sobre este conjunto pode ser verificado na tabela 5.2.

O gráfico 5.13 é o gráfico de seqüência de T2. A partir do próprio gráfico desta tarefa é possível perceber a maior oscilação de seus tempos de resposta. Dados referentes a esta tarefas são listados na tabela 5.2.

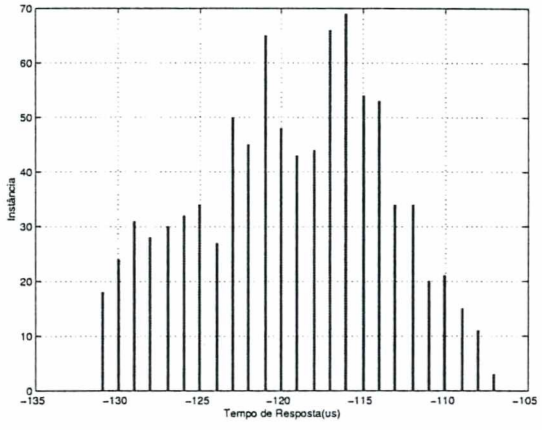


Figura 5.12
Histograma parcial de R - texto-TR7 T1

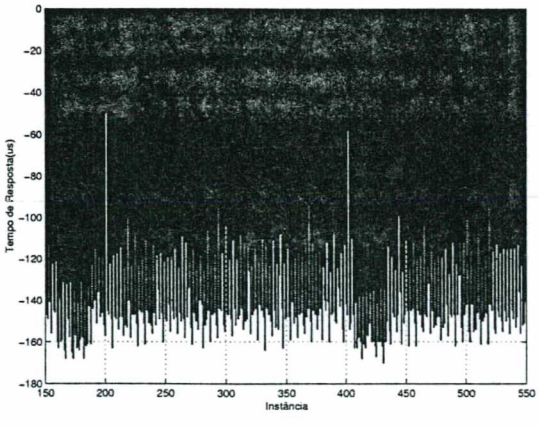


Figura 5.13
Gráfico de R - texto-TR7 T2

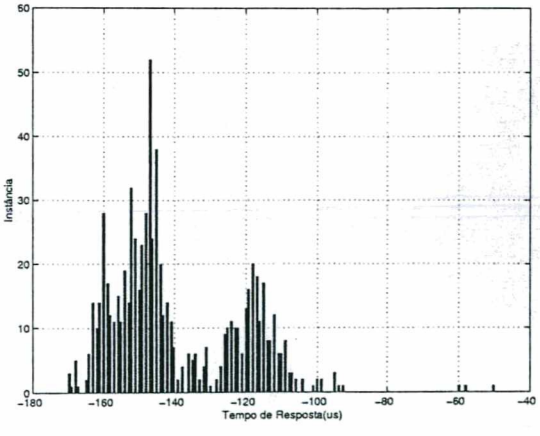


Figura 5.14
Histograma de R - texto-TR7 T2

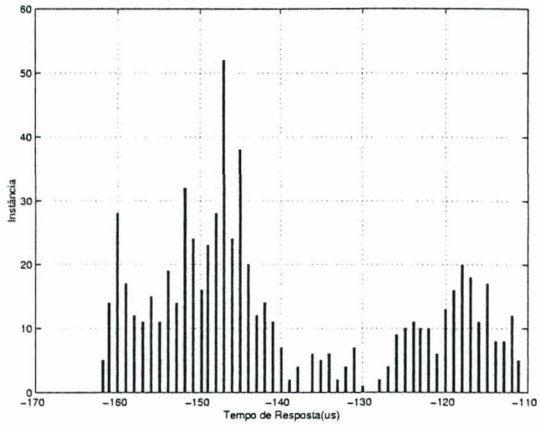


Figura 5.15
Histograma parcial de R - texto-TR7 T2

O gráfico 5.14, ilustra o histograma de T2, que apresenta ocorrências mais distribuídas pelo eixo dos tempos que os histogramas anteriormente descritos. A variação de T2 é a maior apresentada pelas tarefas descritas até o momento e resulta em um histograma com curva diferente da curva das duas tarefas anteriores da mesma aplicação.

A curva descrita pelo histograma de T2 não apresenta formato de curva de distribuição normal, existem neste gráfico, duas concentrações maiores de ocorrências de instâncias. No histograma do gráfico 5.15, construído a partir de 90% das instâncias com tempo de resposta mais distantes dos valores extremos, a cauda do histograma desaparece.

A quarta tarefa de tempo real do cenário, tarefa T3 tem período de 250ms e prioridade imediatamente inferior à prioridade de T2. Conforme ilustrado no gráfico 5.16, a seqüência de tempos de resposta das instancias de T3 descrevem também um gráfico uniforme, não muito diferente dos gráficos de tarefas anteriormente descritas.

O gráfico de distribuição parcial descrito pelo histograma 5.18 é obtido pelos tempos de resposta das instâncias de T3. Neste caso, o histograma parcial não apresenta grande diferença em relação ao histograma do gráfico 5.17 com o número total de instâncias.

A quinta tarefa do cenário texto-TR7, a tarefa de menor valor de prioridade, possui período igual a 350ms. Na sessão de experimento foi realizada a execução de 500 instâncias desta tarefa, 400 dos quais são ilustrados no gráfico 5.19.

Conforme indicado na tabela 5.2, a tarefa T4 é a que mais apresenta oscilações nos tempos de resposta neste cenário. O gráfico de seqüência descrito por esta tarefa permaneceu dentro do padrão estabelecido pelo resto da aplicação.

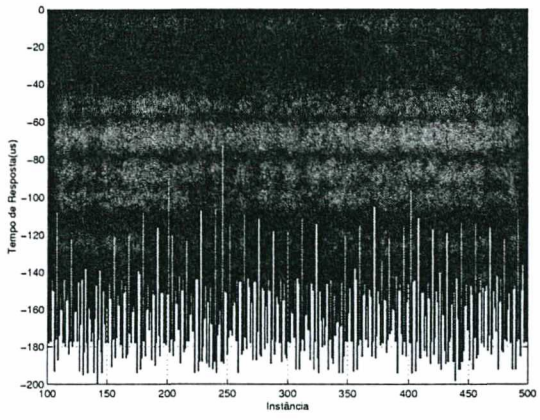


Figura 5.16
Gráfico de R - texto-TR7 T3

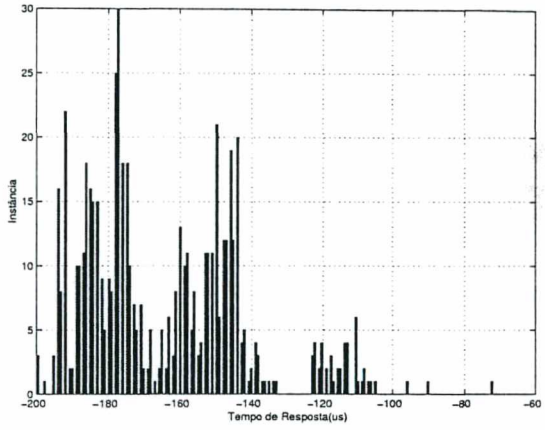


Figura 5.17
Histograma de R - texto-TR7 T3

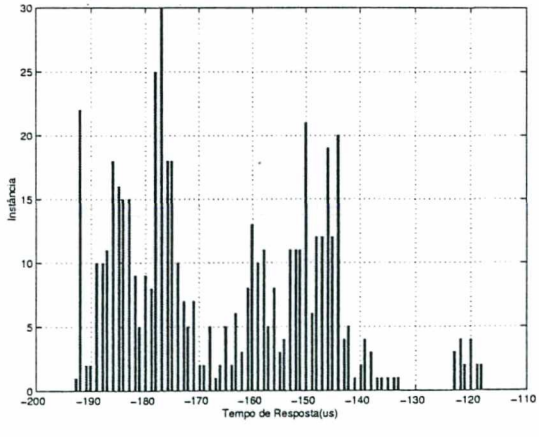


Figura 5.18
Histograma parcial de R - texto-TR7 T3

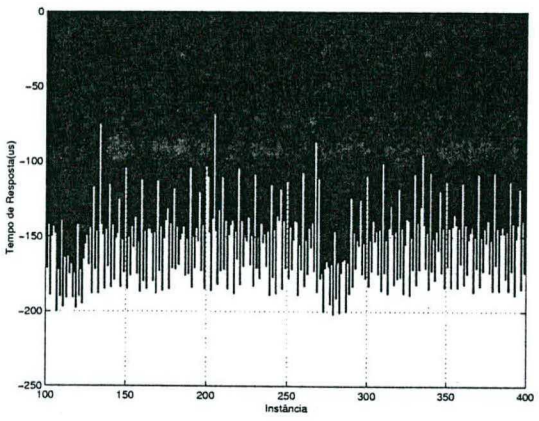


Figura 5.19
Gráfico de R - texto-TR7 T4

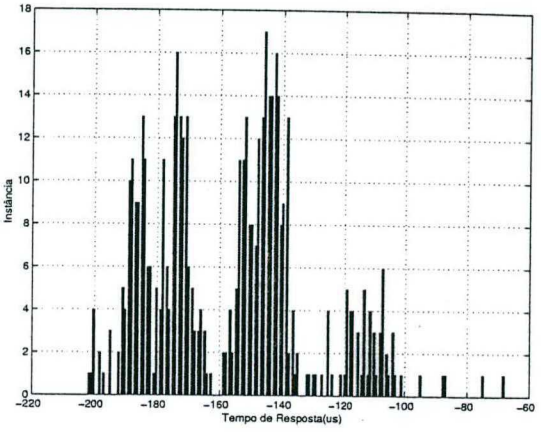


Figura 5.20
Histograma de R - texto-TR7 T4

O gráfico 5.20 ilustra o histograma de T4. Pelo histograma parcial de T4 na figura 5.21 é verificado que esta tarefa apresenta grupos diferentes de instâncias com tempos de resposta próximos de determinados valores de tempo, provocando a separação do histograma em picos.

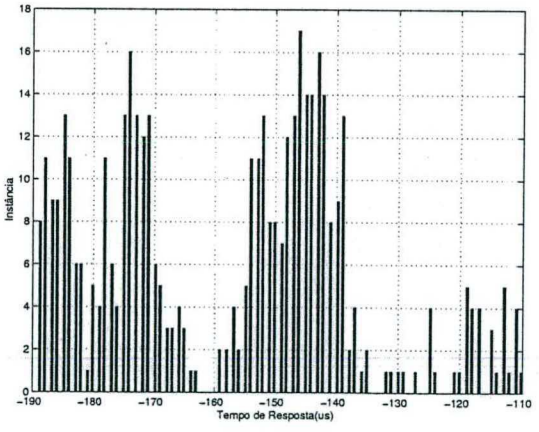


Figura 5.21
Histograma parcial de R - texto-TR7 T4

A maior distribuição pelo eixo do tempo ilustrado no histograma de T4 não apresenta, como T1 e T2, a ocorrência de poucas ativações com tempo de resposta distantes do tempo de resposta típico. Por isso, não é caracterizado no histograma desta tarefa uma cauda longa, com valores pouco atingidos.

No histograma parcial de T4, construído com 90% dos valores mais próximos de um

valor médio de R não difere muito do histograma 5.20.

O cenário texto-TR7 foi elaborado com o objetivo de observar o comportamento de aplicações de tempo real no Linux executando no sistema supostamente dedicado a uma aplicação com múltiplas tarefas. Excetuando os processos do próprio sistema operacional, cada tarefa de tempo real estaria sujeita apenas a interferências provocadas por outras tarefas da mesma aplicação.

Experimentos realizados neste cenário mostraram que cada tarefa periódica estabelece, de acordo com suas características individuais, um padrão quanto aos tempos de resposta de suas instâncias. Isto é, existe para cada tarefa, a repetição do comportamento (baseado no tempo de resposta) a cada conjunto de instâncias executadas. O número de instâncias que compõe esta conjunto é dependente do período da tarefa.

A execução individual de cada uma das tarefas sob as mesmas condições estabelecidas para as cinco tarefas juntas, apresenta tempos de resposta com menor variação entre eles. Este fato é constatado até mesmo para a tarefa T0, que possui a maior prioridade do conjunto. O aumento da variação dos tempos de resposta, entretanto, é desprezível tendo em vista a magnitude da faixa de variação apresentados pelos mesmos (dezenas de microsegundos).

A execução das tarefas de tempo real na mesma aplicação, porém com a exclusão das tarefas comuns, de leitura e escrita em dispositivos, resultou em experimentos sem diferenças significativas em relação ao cenário texto-TR7.

No cenário gui-TR7, a mesma aplicação é executada a partir de uma janela do ambiente gráfico. Assim, somado às tarefas não tempo real presentes na aplicação, estão todos os processos responsáveis pela manutenção de interface gráfica de usuário.

Sabendo que tais processos executam no sistema sob prioridade de tarefas comuns, os resultados esperados para este cenário seriam dados semelhantes aos obtidos pela análise do cenário texto-TR7.

O gráfico 5.22 ilustra a seqüência de tempos de resposta de 400 instâncias de T0. Pela observação dos gráficos desta seção, tanto pelos gráficos de seqüência quanto pelos graficos de histograma, verifica-se uma grande semelhança entre este cenário e o cenário anterior.

Embora exista, em geral, o aumento da faixa de variação e de desvio padrão dos tempos de resposta das tarefas de TR7 no cenário com interface gráfica de usuário, tais aumentos são pequenos, e não chegam a resultar em gráficos muito diferentes nos dois cenários, tão pouco a apresentar valores que possam levantar características particulares de um determinado cenário. Isto é, eventualmente, uma sessão de experimento no cenário texto-TR7 pode apresentar variação e/ou desvio padrão maiores que gui-TR7.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-83	25	108	8.97	-77	-50	27	6.83
T1	-101	9	110	10.64	-93	-60	33	8.46
T2	-113	17	130	15.53	-105	-58	47	12.61
T3	-128	-36	92	17.95	-121	-63	58	14.55
T4	-126	3	129	17.34	-114	-56	58	13.32

Tabela 5.3: gui-TR7 (Valores em microssegundos)

A tabela 5.3 apresenta os dados relacionados à execução de TR7 neste cenário.

Todas as tarefas deste cenário apresentam gráficos de seqüência bastante semelhantes aos gráficos do cenário texto-TR7. Como pode ser confirmado pelos dados da tabela 5.3, as sessões de experimento ilustradas nesta seção diferem de poucos microssegundos quanto à variações e desvio padrão. Os dados coletados a partir da execução de experimentos nestes dois cenários apresentam resultados próximos mesmo em experimentos de longa duração.

Tanto gráficos de seqüência quanto os histogramas das tarefas, do experimento são semelhantes nos dois cenários. Os gráficos e histogramas correspondentes ao experimento neste cenários são ilustrados pelas figuras 5.22 a 5.36.

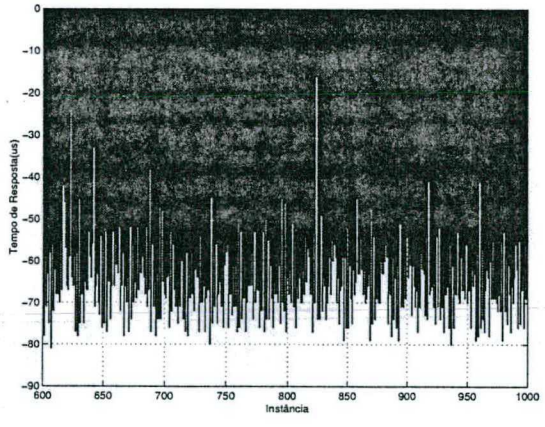


Figura 5.22
Gráfico de R - gui-TR7 T0

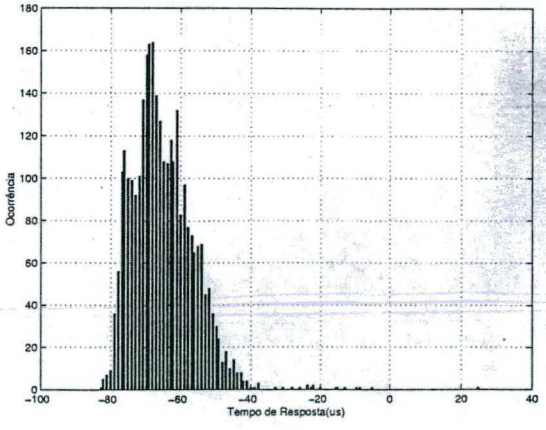


Figura 5.23
Histograma parcial de R - gui-TR7 T0

O cenário gui-TR7 foi elaborado com o objetivo de comparar o comportamento de uma aplicação de tempo real no Linux fazendo uso da interface gráfica de usuário com o comportamento em modo texto. Os experimentos conduzidos permitem afirmar que não existe mudança considerável de comportamento das tarefas de tempo real.

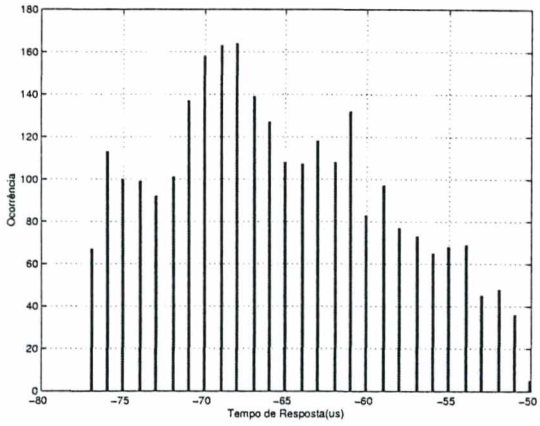


Figura 5.24
Histograma de R - gui-TR7 T0

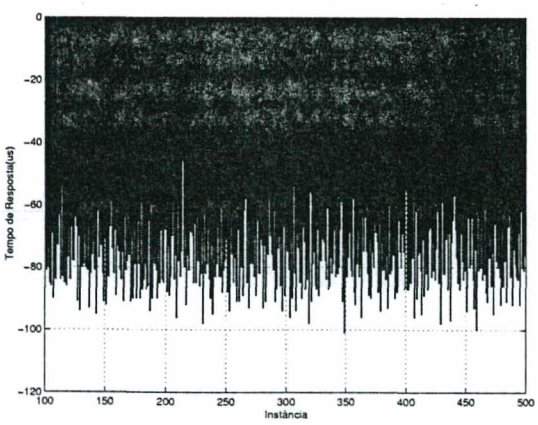


Figura 5.25
Gráfico de R - gui-TR7 T1

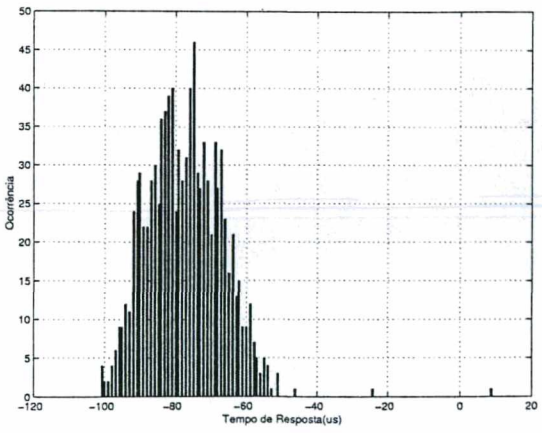


Figura 5.26
Histograma de R - gui-TR7 T1

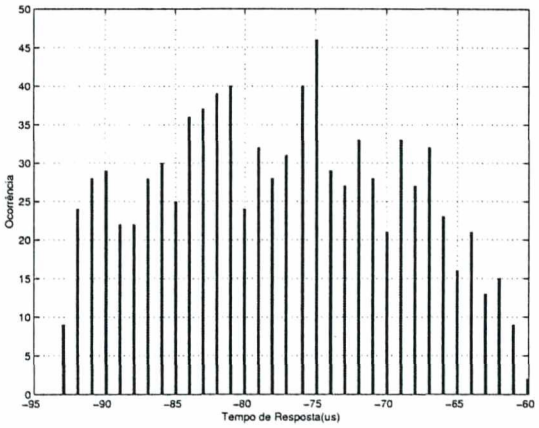


Figura 5.27
Histograma parcial de R - gui-TR7 T1

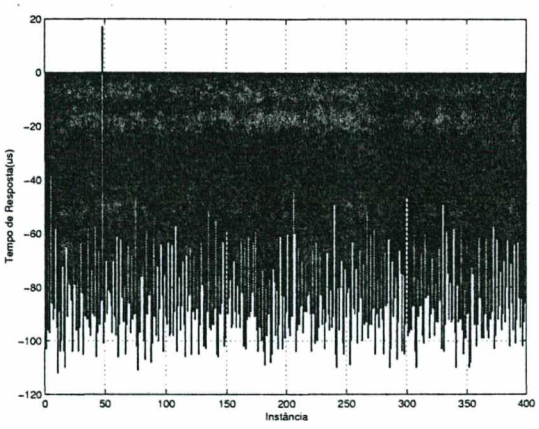


Figura 5.28
Gráfico de R - gui-TR7 T2

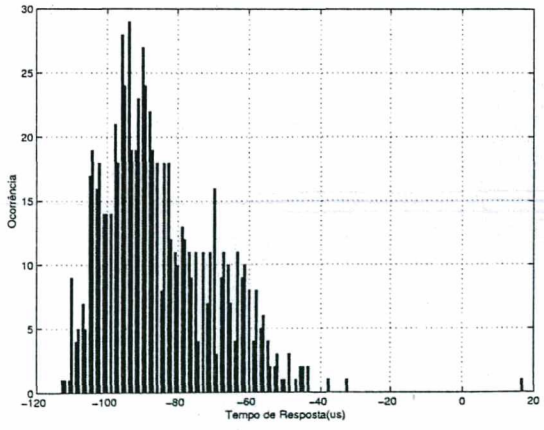


Figura 5.29
Histograma de R - gui-TR7 T2

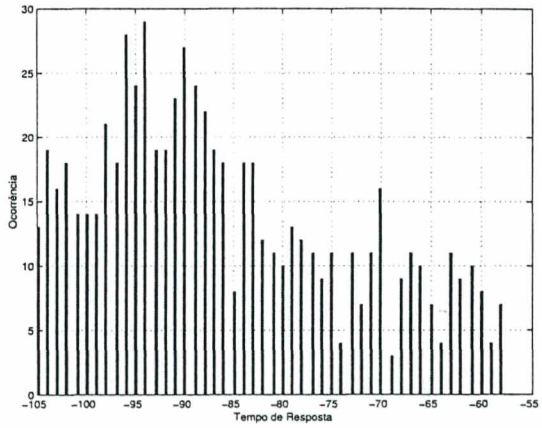


Figura 5.30
Histograma parcial de R - gui-TR7 T2

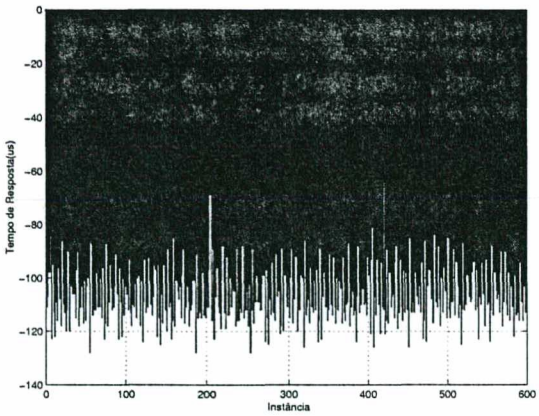


Figura 5.31
Gráfico de R - gui-TR7 T3

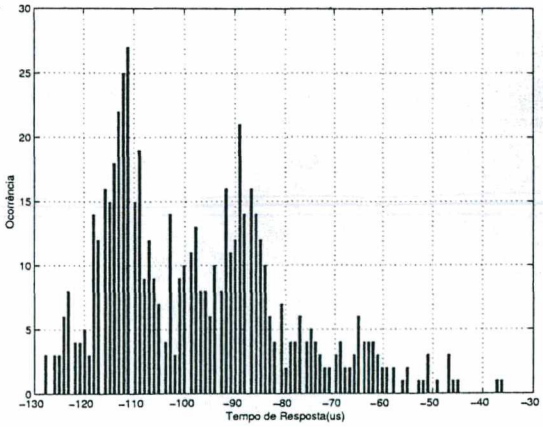


Figura 5.32
Histograma de R - gui-TR7 T3

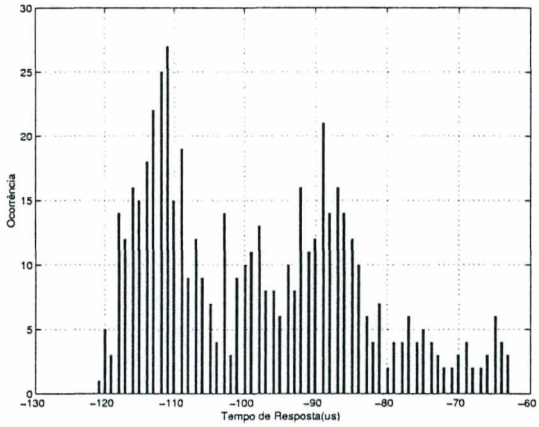


Figura 5.33
Histograma parcial de R - gui-TR7 T3

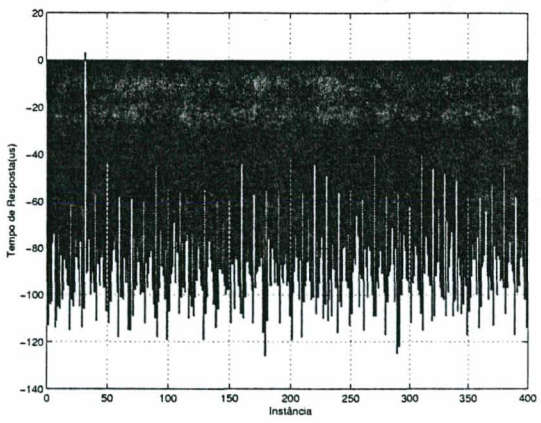


Figura 5.34
Gráfico de R - gui-TR7 T4

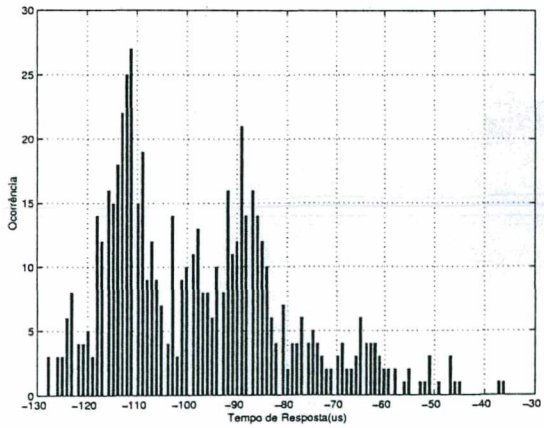


Figura 5.35
Histograma de R - gui-TR7 T4

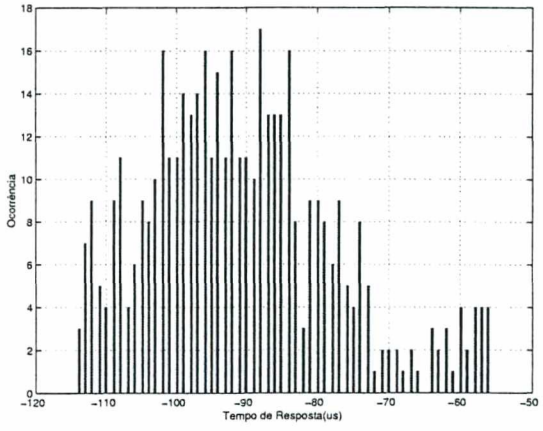


Figura 5.36
Histograma parcial de R - gui-TR7 T4

Experimentos em ambos os cenários (texto-TR7 e gui-TR7) apresentaram eventualmente, picos no gráfico de seqüência de suas tarefas correspondentes a um tempo de resposta muito maior que os demais valores de R apresentados pelo restante das instâncias da mesma tarefa. Este tempo de resposta é geralmente apresentado por uma única instância de cada tarefa.

Devido ao fato da ativação correspondente ao tempo de resposta em questão ocorrer por volta do mesmo instante (em relação ao início do experimento). Conclui-se que a este tempo de resposta estão associadas possíveis interferências provocadas por atividades no kernel relacionadas a manutenção da própria aplicação de tempo real.

Em experimentos em que existe a ocorrência deste valor de tempo de resposta maior, a propagação desta distorção não se estende por muitas instâncias, de forma que o restante do experimento apresenta comportamento uniforme.

A execução da aplicação TR7 descrita nesta seção nos dois cenários são bastante semelhantes. Durante a maior parte do experimento, o comportamento das tarefas é bastante homogêneo. Porém, deve ser considerado a possibilidade de obter tempos de resposta fora do padrão de tempos da tarefa. A ocorrência de perturbações no tempo de resposta das tarefas nestes experimentos ocorre com certa freqüência, e implica em instâncias com tempo de resposta medidos em algumas dezenas de milissegundos.

5.3 Aplicação TR9: cenários texto-TR9 e gui-TR9

Assim como na aplicação TR7, a aplicação TR9 possui *threads* de tempo real e *threads* comuns. Embora as *threads* de tempo real sejam as mesmas, novas *threads* foram adicionadas nesta aplicação. TR9 possui quatro *threads* não tempo real; além das *threads* de teclado e impressão na tela, uma *thread* de manipulação de arquivos em disco e uma *thread* de requisições de pacotes de rede.

O cenário texto-TR9 é composto pela aplicação TR9 em execução no ambiente em modo texto, sem qualquer carga adicional além de processos do próprio Linux. Embora seja bastante semelhante ao cenário texto-TR7, a inclusão de tarefas de utilização de rede e disco provocam grande impacto sobre as tarefas de tempo real. Os tempos de resposta gerados pelos cenários envolvendo TR9 são fornecidos em milissegundos.

As *threads* de acesso a disco e rede de TR9 são tarefas comuns, sem qualquer privilégio de execução, mas exigem mais recursos do sistema que as tarefas de E/S de vídeo e teclado. Foram implementadas com o objetivo de fornecer ao cenário maior número de situações de disputa por recursos e interferência sobre as tarefas de tempo real. Possibilitando verificar o tratamento dado pelo Linux a tarefas de tempo real diante da presença de tarefas comuns utilizando de maneira intensiva alguns recursos do sistema.

No cenário texto-TR9, a execução de T0 resulta num gráfico de seqüência bastante diferente dos cenários anteriormente descritos. Como pode ser verificado pelo gráfico 5.37, não existe um padrão adotado pela seqüência dos tempos de resposta de suas instâncias. Embora grande parte destas obtenham tempo de resposta dentro de uma faixa de dezenas de milissegundos, a variação dos tempos de resposta de T0, considerando todas as suas instâncias é de 77780us.

A variação do tempo de resposta de T0 é maior que o período especificado para esta tarefa. Este fato implica no não cumprimento de restrições temporais de T0 por parte de alguma de suas instâncias. Porém ao considerar somente os 90% das instâncias desta tarefa, obtém-se uma variação de 25 599ms, que se encaixa nos requisitos impostos à tarefa T0.

A tabela 5.4 ilustra os dados relacionados a uma sessão de experimento com TR9 neste cenário. Mesmo existindo uma diminuição dos valores quando somente uma parte das instâncias é considerada, em comparação com o cenário texto-TR7 estes valores são muito mais altos (em texto-TR7, T0 tem tempos de resposta dentro de uma faixa de dezenas de microssegundos).

De acordo com o histograma de T0 ilustrado no gráfico 5.38, a distribuição de tempos de resposta se concentra sobre dois conjuntos de valores, formando dois picos no histogra-

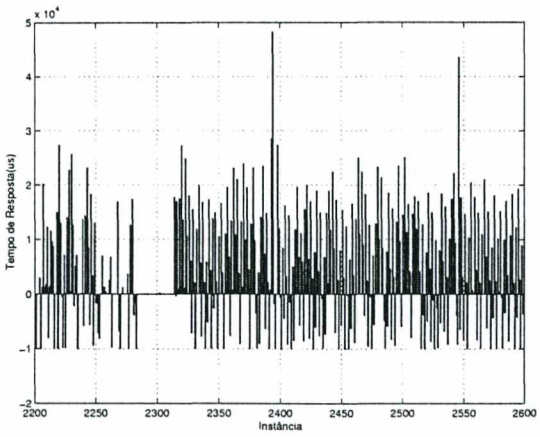


Figura 5.37
Gráfico de R - texto-TR9 T0

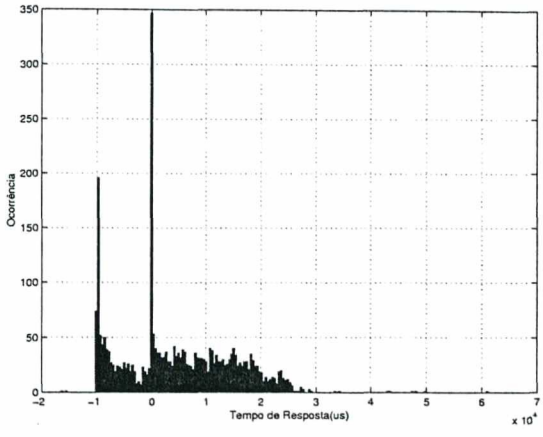


Figura 5.38
Histograma de R -texto-TR9 T0

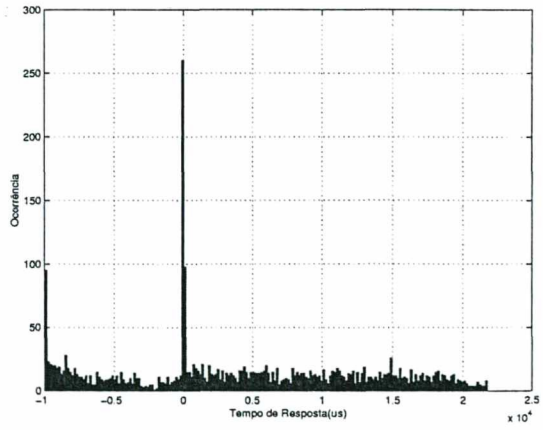


Figura 5.39
Histograma parcial de R - texto-TR9 T0

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-16 611	61 169	77 780	10 154.96	-9 980	25 599	31 769	8 796.65
T1	-10 052	48 277	58 329	10 851.42	-9 985	25 599	35 584	9 349.89
T2	-10 076	49 927	60 003	10 606.88	-10 013	23 376	33 389	8 937.13
T3	-10 095	29 987	40 082	11 235.67	-10 035	26 306	36 341	9 923.07
T4	-10 118	48 265	58 383	11 000.32	-10 029	25 153	35 182	9 316.86

Tabela 5.4: texto-TR9 (Valores em microssegundos)

ma(figura 5.38). Os picos têm entre si aproximadamente 10 000us. Considerando a resolução do timer do Linux (10ms), é razoável supor que se trata do erro causado pelo atraso de um*tick* no *sleep*.

O que pode ser observado pelo histograma é que T0 conclui sua execução em um tempo t ou t+10ms. Embora a curva do histograma 5.38 seja parecida com outras curvas de gráficos já ilustrados neste capitulo, o intervalo de 10 ms é bastante importante quando se trata de tarefas em execução no Linux. Interrupções de tempo geradas com esse período determinam a execução de várias atividades do sistema. Tais atividades, tem prioridade sobre processos executados em espaço de usuário, e portanto provocarão interferência significativa sobre eles, mesmo quando os processos de usuário são tarefas de tempo real.

Este comportamento ilustrado no histograma de T0 é freqüentemente repetido em experimentos que envolvem a execução de tarefas não tempo real no cenário. A interferência provocada por estas tarefas nas tarefas de tempo real tem reflexo direto sobre os tempos de resposta gerados pelas mesmas. O histograma parcial desta tarefa é mostrado na figura 5.39.

Neste cenário as tarefas passam a apresentar tempos de IF maiores que em cenários anteriores. Considerando o numero total de instâncias do experimento, com todas as tarefas, três instâncias apresentaram tempo de IF próximas de 30 milissegundos. As ocorrências destes tempos em intervalos de IF são bastante raras dentro de uma sessão, não estão necessariamente relacionada a uma tarefa ou instante de tempo em particular, mas apresentam o valor de tempo bastante parecidas em todas as suas ocorrências.

A segunda tarefa da aplicação TR9, apresenta um gráfico de seqüência também não uniforme (gráfico 5.40). Conforme indicado na tabela 5.4, a tarefa T1 apresenta uma faixa de variação nos tempos de resposta de suas instâncias de 58 329us. Porém, esta tarefa apresenta poucas de suas instâncias com tempo de resposta próximas do limite superior. De fato, ao excluir 10% dos tempos de resposta próximos aos valores limite da própria tarefa, a faixa de variação passa a ser de 35 584us (tabela 5.4).

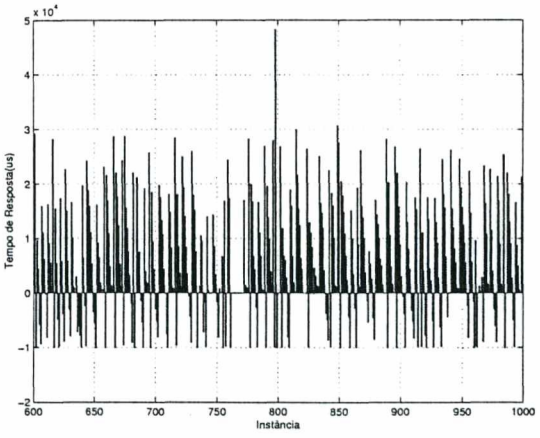


Figura 5.40
Gráfico de R - texto-TR9 T1

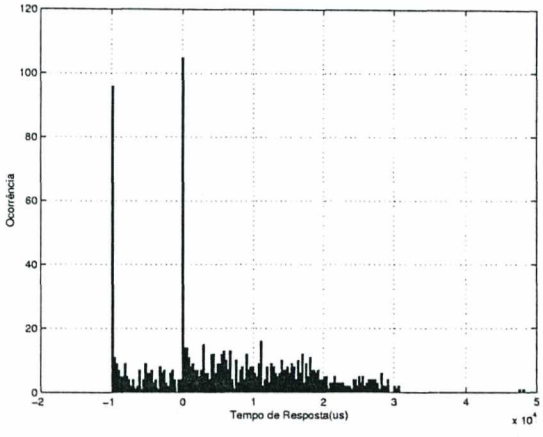


Figura 5.41
Histograma de R -texto-TR9 T1

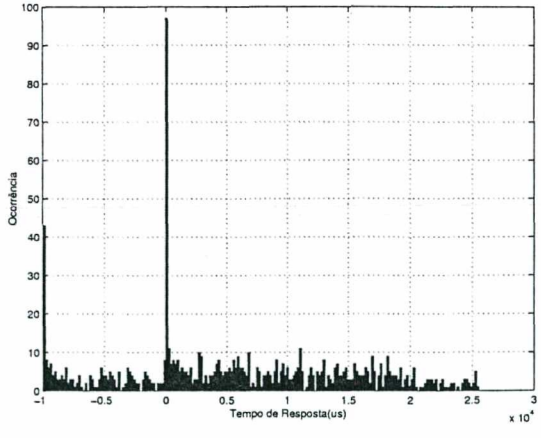


Figura 5.42
Histograma parcial de R -texto-TR9 T1

O histograma de T1, no gráfico 5.41 ilustra uma curva parecida com a curva descrita pelo histograma de T0. As instâncias desta tarefa também se distribuem, de acordo com seus tempos de resposta, em duas maiores ocorrências. Como pode ser confirmado pelo histograma parcial de T1(gráfico 5.42), o maior conjunto de ocorrências se separam no gráfico por aproximadamente 10ms, assim como descrito em T0.

Quanto ao tempo gasto pela tarefa entre o início e a conclusão de sua execução, T1 também apresenta valores mais altos que nos demais cenários já descritos. Novamente é constatado que tempos de IF de pior caso não correspondem a tempos de R de pior caso, pois não incluem o *jitter* de chegada da tarefa.

Assim como observado na descrição das tarefas anteriores, o gráfico de seqüência da tarefa T2 (figura 5.43) não apresenta um padrão. Embora a maior parte de suas instâncias esteja contida numa faixa bem definida de tempo, T2 apresenta picos no início de sua execução correspondentes a tempo de resposta de pior caso para esta tarefa que contribuem para expansão de sua faixa de variação.

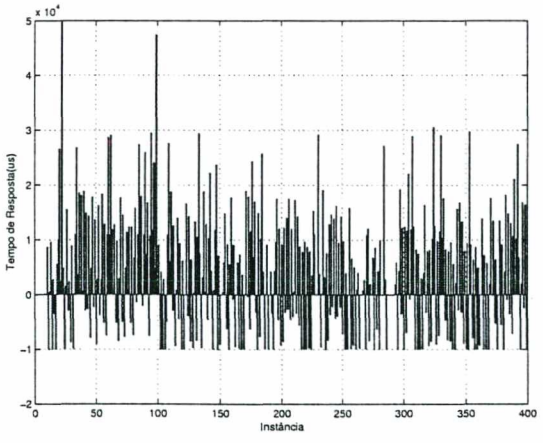


Figura 5.43
Gráfico de R - texto-TR9 T2

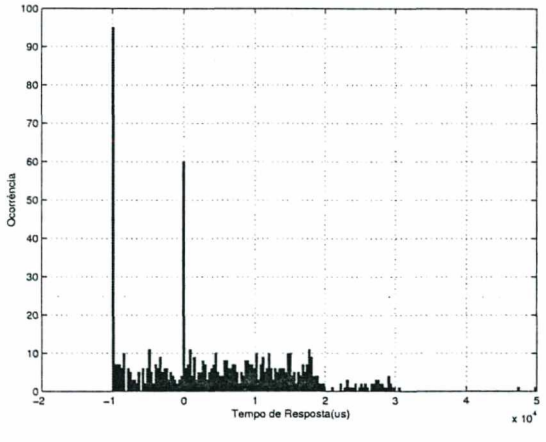


Figura 5.44
Histograma parcial de R -texto-TR9 T2

O histograma de T2 (gráfico 5.44), por sua vez, apresenta uma curva de distribuição parecida com a curva de T0 e T1. O cenário proporciona também a esta tarefa, uma distribuição de ocorrência que se concentra em tempos de resposta separados por um intervalo de 10ms. O histograma parcial de T2, apresentado no gráfico 5.45 confirma a distribuição das ocorrências de T2. Os dois maiores picos do gráfico ocorrem separados pelo intervalo de 10ms.

As tarefas T3 e T4, assim como as demais tarefas, apresentam gráficos semelhantes aos das tarefas anteriormente descritas (gráficos 5.46 a 5.51). Não existe padrão no gráfico de

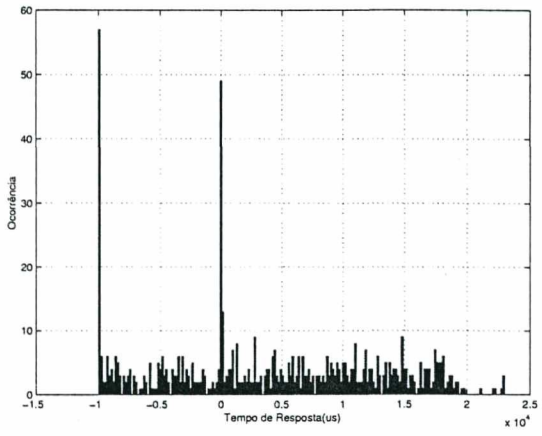


Figura 5.45
Histograma de R - texto-TR9 T2

seqüência destas tarefas, e a curva do histograma das mesmas apresenta o mesmo formato de T0, T1 e T2.

Os dados relacionados a esta tarefa estão ilustrados na tabela 5.4.

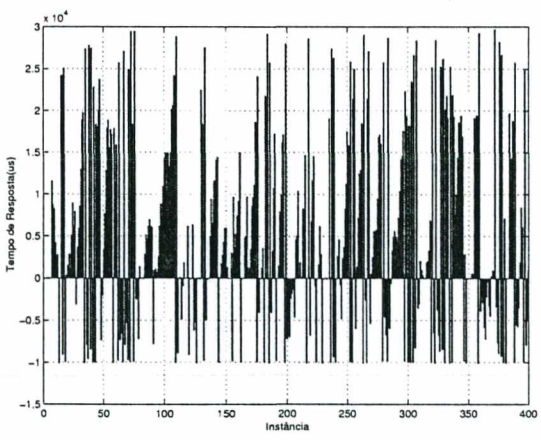


Figura 5.46
Gráfico de R - texto-TR9 T3

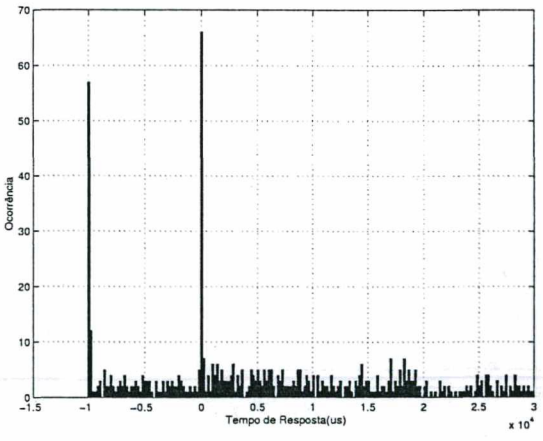


Figura 5.47
Histograma de R -texto-TR9 T3

O tempo gasto por T4 entre o início e a conclusão de sua execução variam na maior parte do experimento em poucos microssegundos. Ainda assim, instâncias com tempo de IF de alguns milissegundos ocorrem na execução desta tarefa.

O cenário texto-TR9, mostra que a inclusão de tarefas comuns que representam adição de carga no sistema na forma de acesso a rede e disco, alteram consideravelmente os tempos

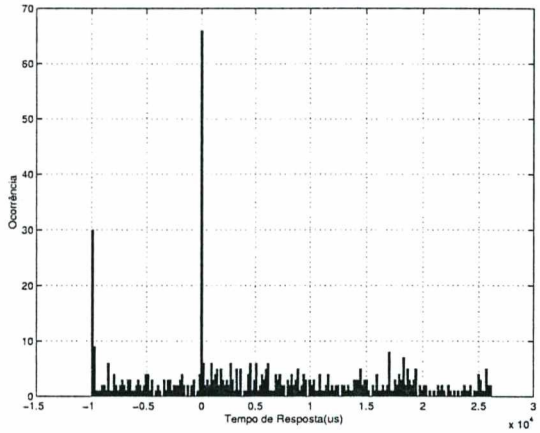


Figura 5.48
Histograma parcial de R -texto-TR9 T3

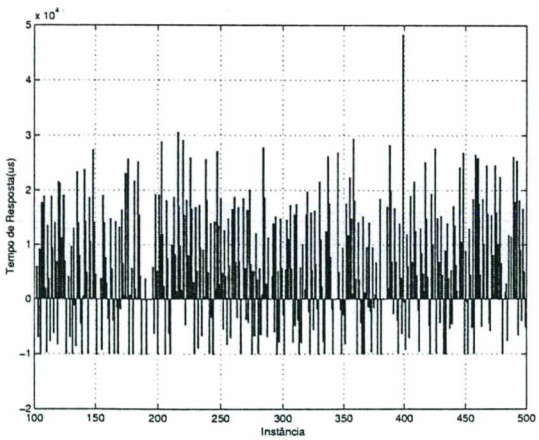


Figura 5.49
Gráfico de R - texto-TR9 T4

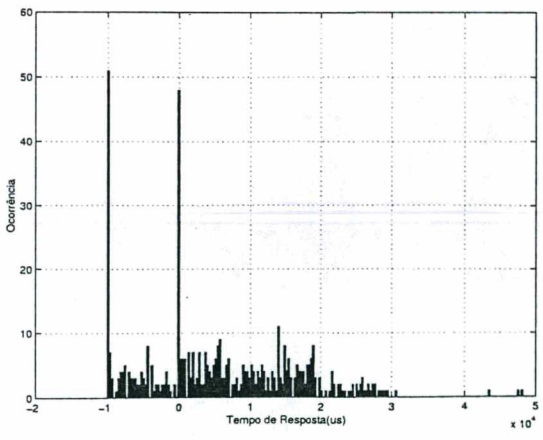


Figura 5.50
Histograma de R -texto-TR9 T4

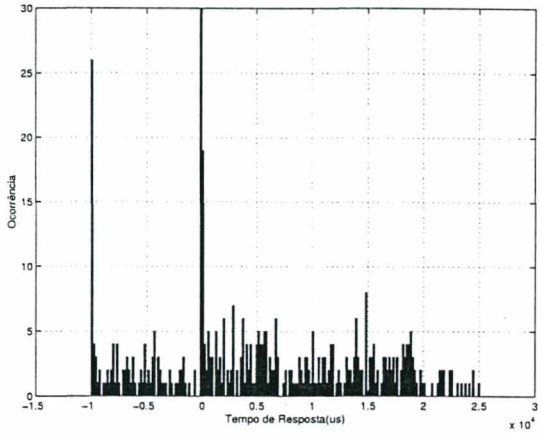


Figura 5.51
Histograma parcial de R - texto-TR9 T4

de resposta das tarefas de tempo real.

Mesmo sendo as tarefas adicionadas a este cenário, tarefas com prioridade menores que as tarefas de tempo real, são tarefas que exigem recursos que ativam processos dentro do kernel, que por sua vez tem prioridade maior que tarefas de tempo real no espaço do usuário e provocam interferência sobre as mesmas.

Neste cenário são percebidas as primeiras ocorrências de interferências dentro do intervalo de tempo medido entre o início e a conclusão da tarefa de tempo real. Intervalos com valores medidos por dezenas de milissegundos são freqüentemente encontrados nas sessões de experimento. Tais valores nem sempre correspondem às mesmas ativações com os maiores valores de R, e ocorrem com baixa freqüência em qualquer das tarefas (em menos de 1% das instâncias).

O experimento executado no cenário texto-TR9 confirma o comportamento do Linux, como de um sistema voltado para propósitos gerais, que permite a interferência de tarefas comuns sobre tarefas de tempo real.

O cenário gui-TR9, assim como o texto-TR9, apresenta gráficos de seqüência sem um padrão definido. O que pode ser observado do comportamento de tarefas de tempo real neste cenário é que não existem grandes diferenças nos tempos de resposta no que diz respeito à presença ou não de interface gráfica de usuário carregada no sistema. A grande contribuição dos cenários compostos pela aplicação TR9 reside na verificação de interferências provocadas pelas *threads* não tempo real. No caso desta aplicação, a interferência provocada pela presença de interface gráfica se dissolve em valores maiores, tornando os resultados dos experimentos deste cenário bastante parecidos com os do cenário anterior.

O gráfico de seqüência de T0, gráfico 5.52, ilustra o tempo de resposta de 400 instâncias de T0. Assim como no cenário anterior, T0 não adota um padrão quanto aos tempos de resposta da seqüência de suas instâncias. A variação dos tempos de resposta, desvio padrão, R mínimo e máximo para o total e para 90% das instâncias desta tarefa são ilustrados na tabela5.5. Pelos valores que indicam a variação de tempo de resposta desta tarefa, verifica-se que não pode ser garantida a especificação de deadline igual ao período. A variação dos tempos de resposta é maior que 60ms, enquanto o período estabelecido para a mesma é de 50ms.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-17 993	42 731	60 724	10 590.15	-9 971	24 166	34 137	9 152.95
T1	-9 989	34 762	44 751	10 755.90	-9 950	25 254	35 204	9 377.31
T2	-10 021	32 588	42 608	10 657.71	9 978	22 405	32 383	9 353.68
T3	-10 036	56 246	66 282	10 467.43	-9 992	22 403	32 395	8 864.27
T4	-10 051	34 322	44 373	11 050.36	-9 971	26 217	36 188	9 640.92

Tabela 5.5: gui-TR9 (Valores em microssegundos)

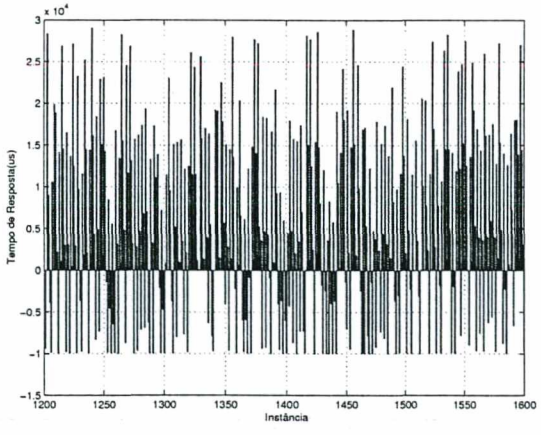


Figura 5.52
Gráfico de R - gui-TR9 T0

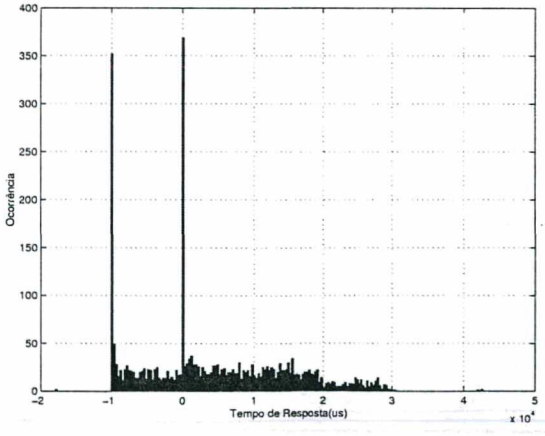


Figura 5.53
Histograma de R -gui-TR9 T0

A curva descrita pelo histograma de T0 (gráfico 5.53) neste cenário é semelhante à curva descrita pela mesma tarefa em texto-TR9. A distribuição de ocorrências de tempo de resposta compõem dois picos no histograma, assim como observado no cenário anterior. Estes picos se distanciam no gráfico por 10ms. O histograma parcial de T0, construído a partir de 90% das instâncias é ilustrado no gráfico 5.54.

O gráfico de seqüência parcial de TR1 é ilustrado pela figura 5.55. Os valores que de-

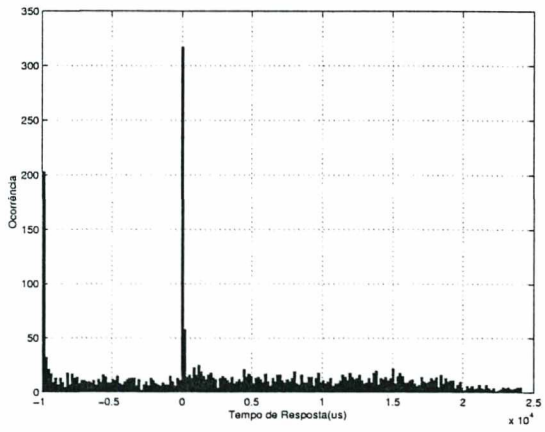


Figura 5.54
Histograma parcial de R - gui-TR9 T0

limitam o domínio de tempos de resposta desta tarefa, a variação, e desvio padrão tanto para o número total de instâncias quanto para 90% das mesmas é dado pela tabela5.5. Os histogramas da tarefa aparecem nas figuras 5.56 e 5.57.

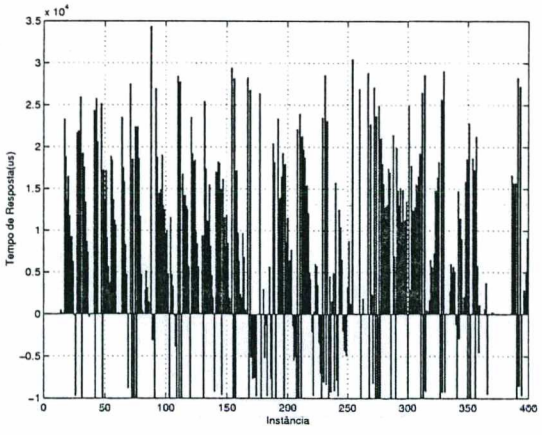


Figura 5.55
Gráfico de R - gui-TR9 T1

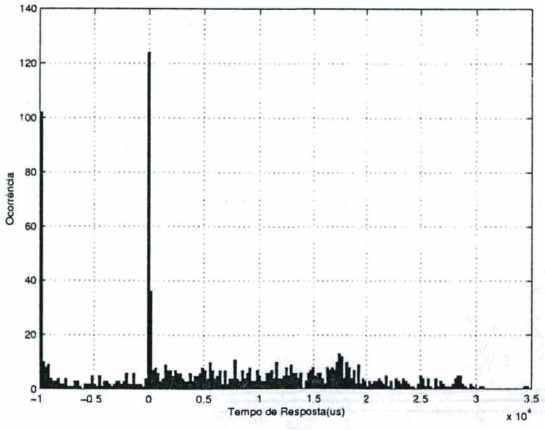


Figura 5.56
Histograma de R -gui-TR9 T1

Verifica-se que não existe nesta tarefa, uniformidade quanto aos valores de R apresentados pelas instâncias de T1. Entretanto, a variação e desvio padrão apresentados pela tarefa são bastante reduzidos quando é considerado somente 90% das instâncias cujo tempo de resposta é mais próximo do valor médio (ver tabela5.5).

As instâncias 200 a 600 da tarefa T2 têm seus tempos de resposta representados no gráfico 5.58.

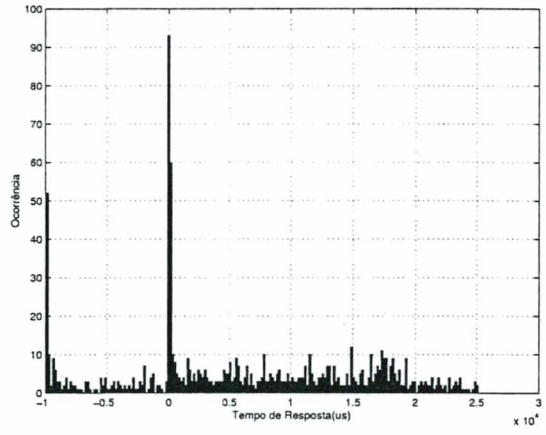


Figura 5.57
Histograma de R -gui-TR9 T1

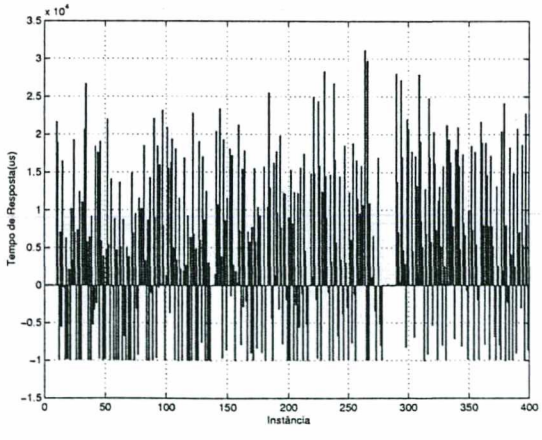


Figura 5.58
Gráfico de R - gui-TR9 T2

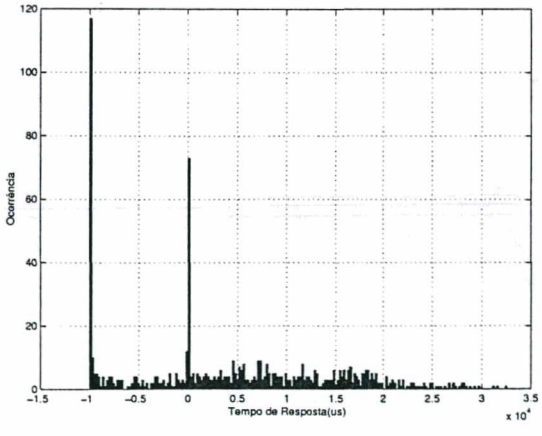


Figura 5.59
Histograma de R -gui-TR9 T2

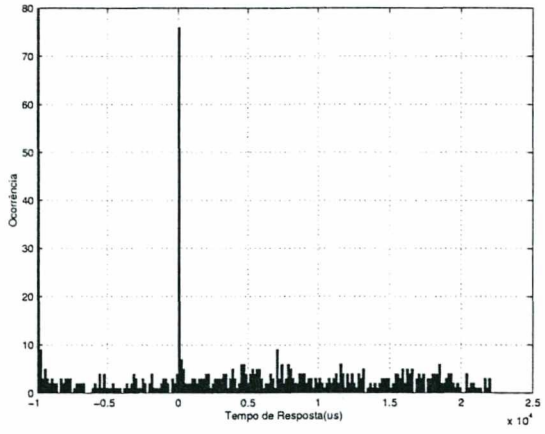


Figura 5.60
Histograma parcial de R - gui-TR9 T2

A distribuição de ocorrências de T2 é representada pelo histograma desta tarefa no gráfico 5.59 e descreve a mesma curva de distribuição de T0 e T1; dois picos maiores indicando um maior número de ocorrências com tempo de resposta em torno de dois valores que se distanciam por 10ms. O histograma de T2 considerando apenas 90% de suas instâncias é ilustrado em 5.60 e permanece bastante parecido com o gráfico 5.59. De modo geral, o que ocorre durante toda a sessão de experimento é o comportamento idêntico de todas as 5 tarefas de tempo real da aplicação. Isto se reflete nos histogramas dos mesmos. Diferenças entre as tarefas passam a ser sutis, indicadas somente pelos valores da tabela5.5.

A tarefa T3, cujo gráfico de seqüência parcial é ilustrado em 5.61 tem comportamento idêntico ao comportamento apresentado em texto-TR9. O histograma de T3 é ilustrado no gráfico 5.62. Neste histograma é observado a mesma curva descrita pelas tarefas anteriormente descritas. O mesmo comportamento é ilustrado no gráfico 5.63 que mostra o histograma parcial de T3.

O gráfico 5.64, ilustra o gráfico de seqüência de 400 instâncias da tarefa T4. Assim como ocorrido com as demais tarefas desta aplicação neste cenário, T4 apresenta um gráfico pouco uniforme. A semelhança dos resultados desta aplicação neste cenário se reflete também nos histogramas dos mesmos, apresentados nos gráficos 5.65 e 5.66.

O cenário gui-TR9 tenta descrever o comportamento geral de uma aplicação que contém tarefas de tempo real em conjunto com tarefas não tempo real de acesso a rede e disco, executando num sistema com a interface gráfica de usuário. A conclusão que se chega no fechamento destas sessões de experimento com TR9 é que a interferência provocada pela interface gráfica de usuário sobre esta aplicação é dissolvida na interferência gerada por

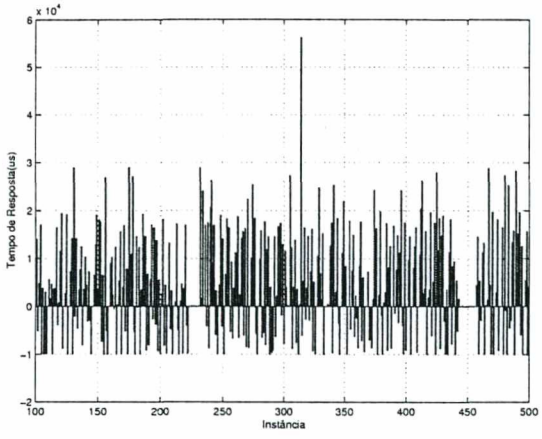


Figura 5.61
Gráfico de R - gui-TR9 T3

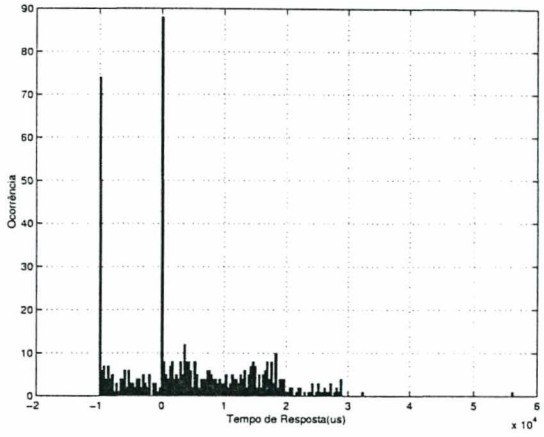


Figura 5.62
Histograma de R -gui-TR9 T3

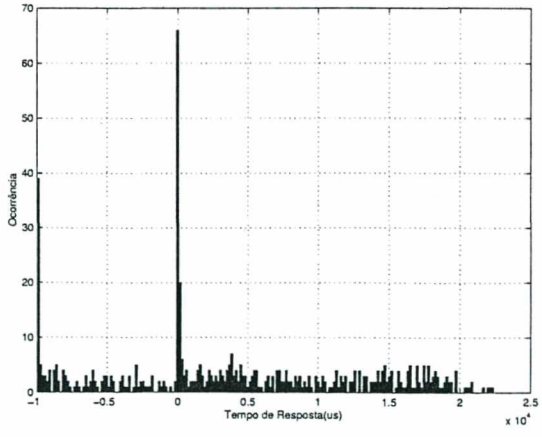


Figura 5.63
Histograma parcial de R -gui-TR9 T3

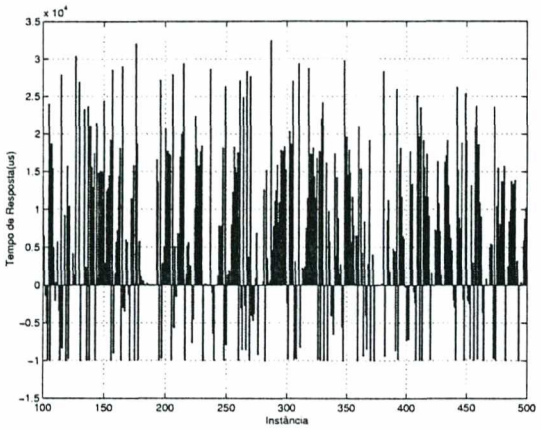


Figura 5.64
Gráfico de R - gui-TR9 T4

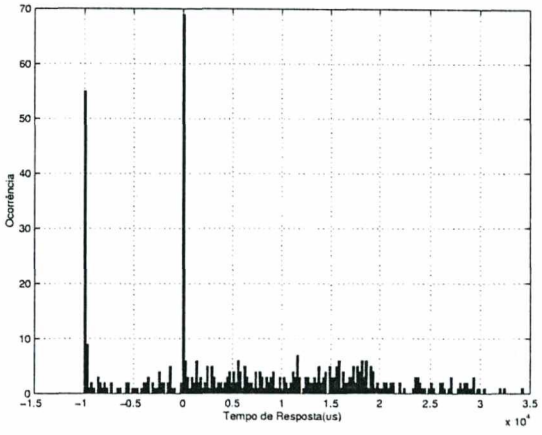


Figura 5.65
Histograma de R -gui-TR9 T4

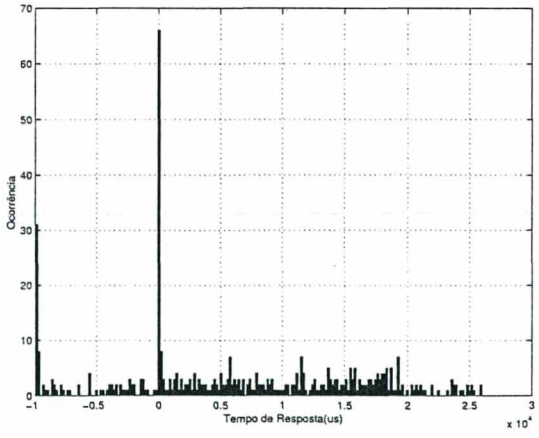


Figura 5.66
Histograma parcial de R - gui-TR9 T4

tarefas que acionam dispositivos de forma intensa.

A interferência provocada pela interface gráfica de usuário observada nos experimentos nos cenários texto-TR7 e gui-TR7 (ou mesmo em texto-TR1 e gui-TR1) tem menor impacto nos cenários que envolvem tarefas não tempo real mais complexas.

A variação de tempo de resposta de T0, que é a tarefa de maior prioridade, nem sempre apresenta o menor valor. Devido ao maior número de ativações desta tarefa em cada sessão, T0 tem maior probabilidade de sofrer atrasos e interferências em meio às suas ativações. O pior caso de latência observado quando executando tarefas de tempo real na presença de outras tarefas não tempo real não pode ser controlado estabelecendo período de cada tarefa sob as condições deste cenário. Os atrasos sofridos pela tarefa são originados pela execução de atividades do kernel, que interferem sobre qualquer tarefa no sistema independente de sua prioridade.

As demais tarefas da aplicação apresentam variação menor que o período estabelecido para as mesmas. Como indicado pela tabela5.5, ao considerar 90% das instâncias das tarefas, o intervalo de variação de tempo de resposta das tarefas do cenário são reduzidos e se tornam bastante próximos.

5.4 Cenários com carga adicional

Os cenários descritos nesta seção são compostos por uma aplicação de tempo real executando em sistema carregado com interface gráfica de usuário e carga adicional. A carga adicional, detalhada no capítulo anterior, foi inserida com o propósito de permitir a observação do comportamento das tarefas de tempo real no Linux, em um ambiente que executa com outras aplicações comuns.

Nesta seção será descrito o resultado obtido pela execução das aplicações TR1, TR7 e TR9 nos cenários com carga. Os gráficos apresentados aqui, foram construídos como na seções anteriores; gráficos de seqüência com 400 instâncias da tarefa, e histogramas totais e parciais de algumas das tarefas de tempo real.

Em cenários anteriores, a aplicação TR1 obteve os menores tempos de resposta dentre todas as aplicações. O tempo de resposta médio desta aplicação foi apresentado em microssegundos tanto no cenário texto-TR1 quanto em gui-TR1.

Na presença de carga adicional, o tempo de resposta desta aplicação sofre aumento considerável, e passa a ser medido em milissegundos. Os valores referentes ao experimento deste cenário são ilustrados na tabela 5.6.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-10 732	24 214	34 946	2 986.69	-9 410	3 114	12 524	876.40

Tabela 5.6: gui-TR1carga (Valores em microssegundos)

Na comparação do gráfico de seqüência 5.67 com os gráficos da seção 5.1 verifica-se a mudança de comportamento da tarefa de TR1 de acordo com o ambiente de execução. Neste gráfico estão ilustradas as relações tempo de resposta e instância das ativações 1000 a 1400.

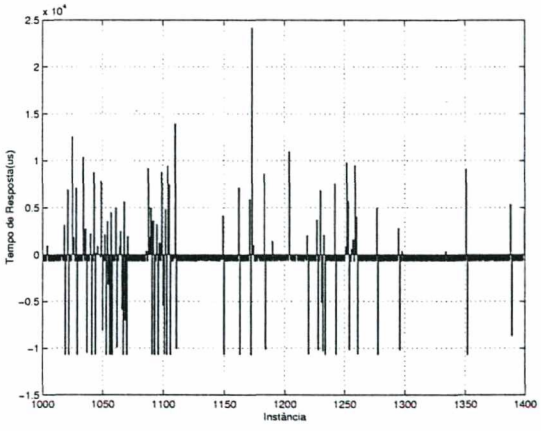


Figura 5.67
Gráfico de R - gui-TR1carga

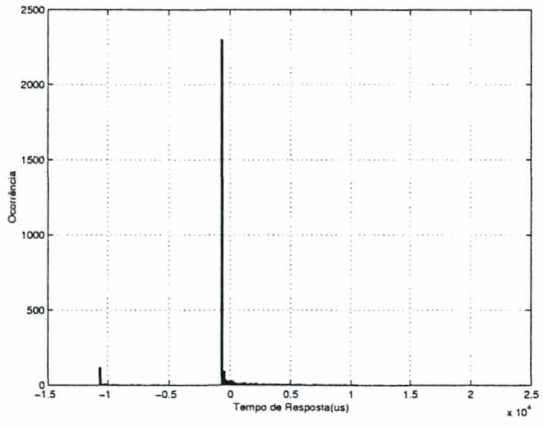


Figura 5.68
Histograma de R - gui-TR1carga

Sessões com variação de tempo de resposta de aproximadamente 70ms foram executadas (pior variação encontrada nos experimentos deste cenário). Porém, instâncias com os tempos de resposta mais altos dentro de cada sessão constituem uma porcentagem baixa de ocorrências. Em geral, estas ocorrências são excluídas dos histogramas construídos com 90% das ativações.

O pico do histograma 5.68 concentra a maioria das instâncias da tarefa. E os limites do mesmo, estabelecem a ocorrência de tempos de resposta próximos dos valores máximos e mínimos que não podem ser percebidos neste gráfico dada a proporção de suas ocorrências no contexto geral do experimento.

Ao considerar somente 90% das ativações desta tarefa, têm-se o histograma 5.69, que elimina algumas ocorrências de 5.68.

Ao excluir do número total de instâncias 5% dos valores mais próximos do tempo de resposta mais baixo, e 5% dos valores mais próximos do tempo de resposta mais alto, obtém-

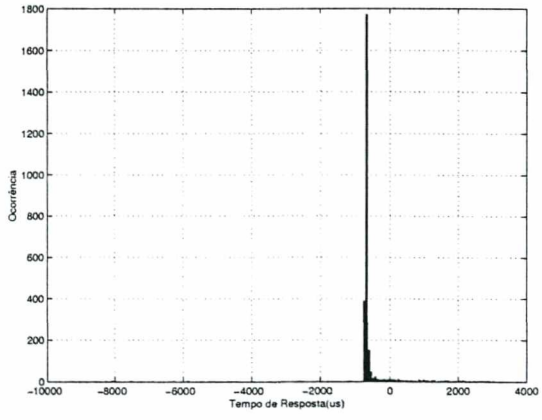


Figura 5.69
Histograma parcial de R - gui-TR1carga

se o histograma do gráfico 5.69. Sob estas condições, o histograma apresenta uma curva com limites estabelecidos numa faixa de 12524us. As concentrações de instâncias, embora pouco evidentes, que apresentam tempo de resposta de 10ms de diferença (os dois picos do gráfico 5.68) não podem ser observadas nesse gráfico.

Quanto ao tempo gasto pela tarefa de tempo real entre o início e o fim de sua execução, grande parte dos intervalos mantêm o tempo em poucos microssegundos. Porém, passam a existir tempos de IF de alguns milissegundos, na sessão em que são baseados os gráficos ilustrados o maior intervalo de IF registrado foi de 2949us.

Como observado no cenário anterior, os picos registrados nos experimentos sem carga adicional com aplicações correspondentes não caracterizam perturbação nos gráficos de seqüência dos cenários desta seção. De maneira análoga a interferência provocada pela presença de interface gráfica de usuário, tais perturbações, quando presentes nos cenários com carga, não são evidentes nos gráficos de seqüência. Os dados gerais sobre o experimento a ser ilustrado nesta seção são indicados na tabela 5.7.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-10 120	18 839	28 959	2 931.17	-8 881	3 148	12 029	1 077.87
T1	-10 168	16 471	26 639	2 965.80	-9 922	3 138	13 060	1 164.76
T2	-10 195	12 219	22 414	3 053.18	-9 819	3 690	13 509	1 250.77
T3	-10 213	12 228	22 441	3 255.72	-10 152	3 780	13 932	1 864.65
T4	-10 244	13 923	24 167	3 184.21	-10 164	3 581	13 745	1 584.73

Tabela 5.7: gui-TR7carga (Valores em microssegundos)

O gráfico 5.70 ilustra relações de instâncias e tempos de resposta correspondentes à tarefa T0 de um experimento com a aplicação TR7 em cenário com carga adicional. Os histogramas aparecem nas figuras 5.71 e 5.72.

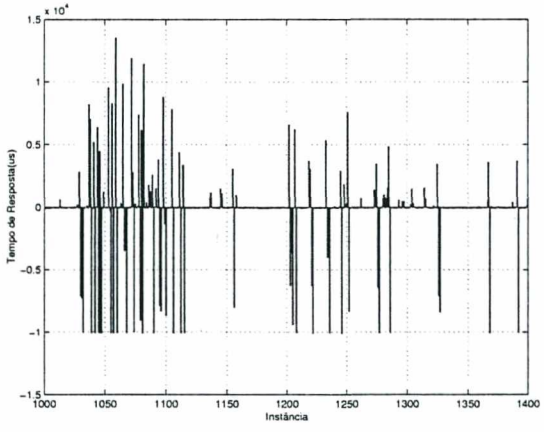


Figura 5.70
Gráfico de R - gui-TR7carga T0

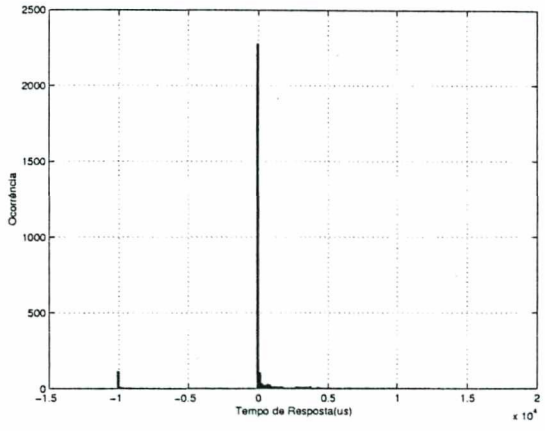


Figura 5.71
Histograma de R -gui-TR7carga T0

Embora grande parte das instâncias desta tarefa variem seus tempos de resposta dentro de uma faixa de dezenas de microssegundos, tempos de resposta de milissegundos também compõem os valores de R de T0. Diferente do ocorrido para esta mesma tarefa nos cenários sem carga, tempos de resposta que chegam a dezenas de milissegundos ocorrem muito mais de uma vez durante a duração de um experimento de 3000 instâncias neste cenário.

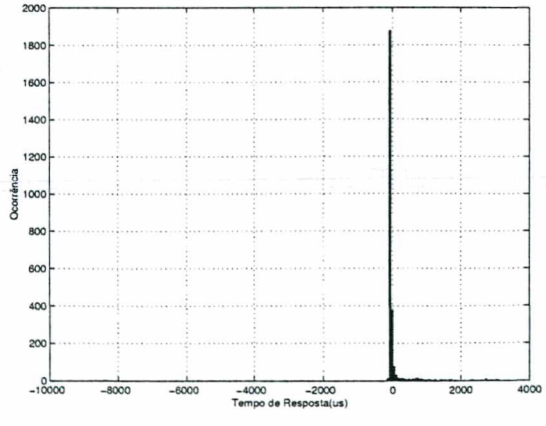


Figura 5.72
Histograma parcial de R - gui-TR7carga T0

As tarefas de tempo real subseqüentes deste cenário, apresentam comportamento análogo ao descrito pela tarefa T0 (figuras 5.73 a 5.84).

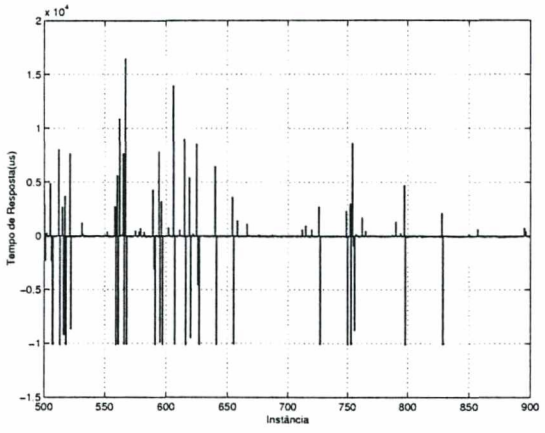


Figura 5.73
Gráfico de R - gui-TR7carga T1

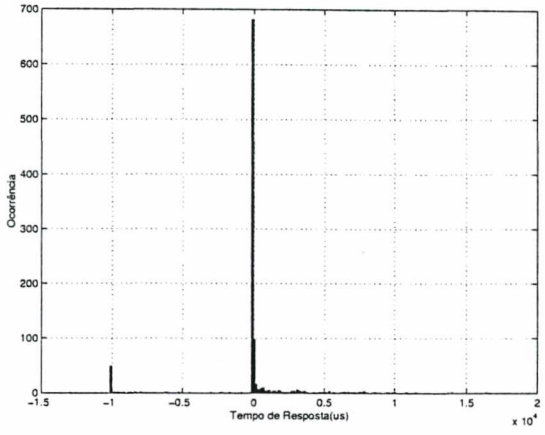


Figura 5.74
Histograma de R -gui-TR7carga T1

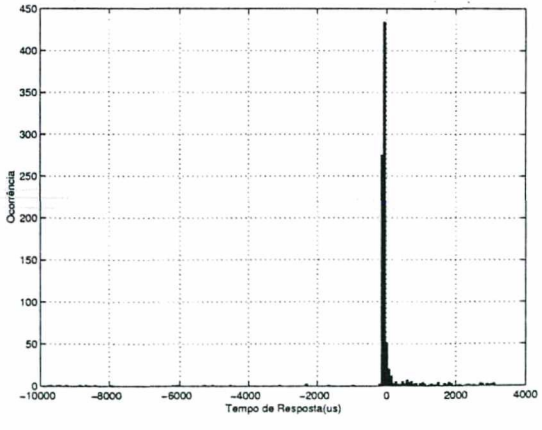


Figura 5.75
Histograma parcial de R -gui-TR7carga T1

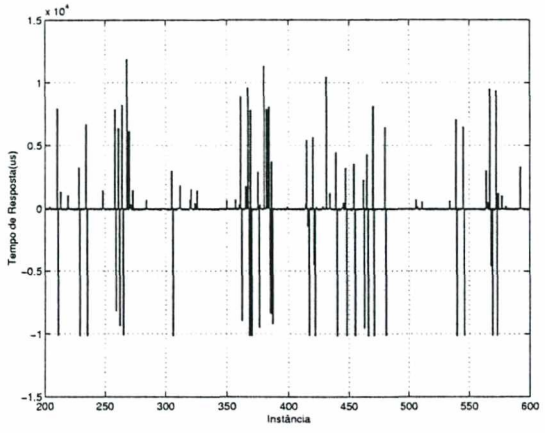


Figura 5.76
Gráfico de R - gui-TR7 T2

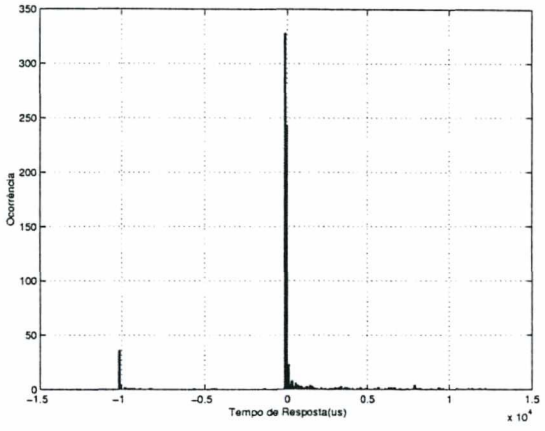


Figura 5.77
Histograma de R -gui-TR7carga T2

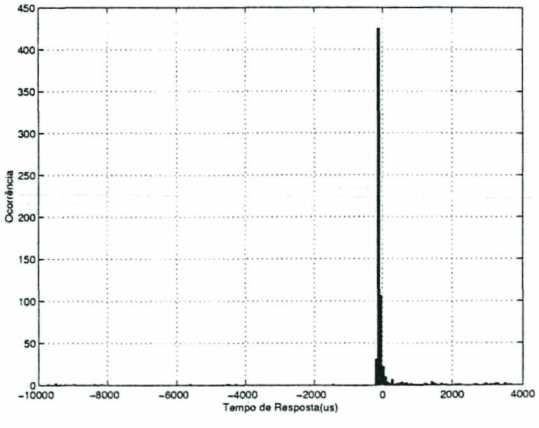


Figura 5.78
Histograma parcial de R - gui-TR9carga T2

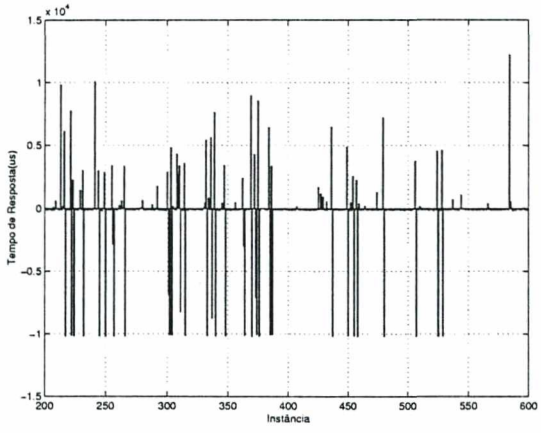


Figura 5.79
Gráfico de R - gui-TR7carga T3

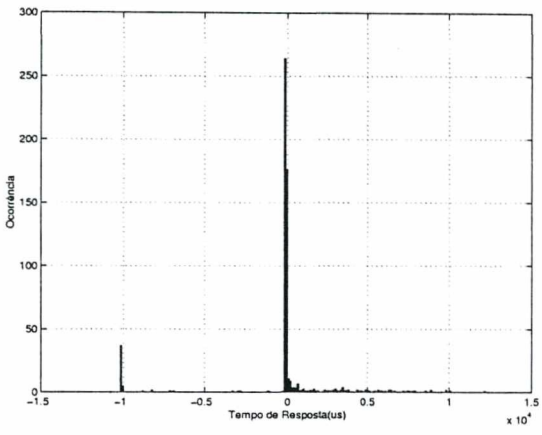


Figura 5.80
Histograma parcial de R -gui-TR7carga T3

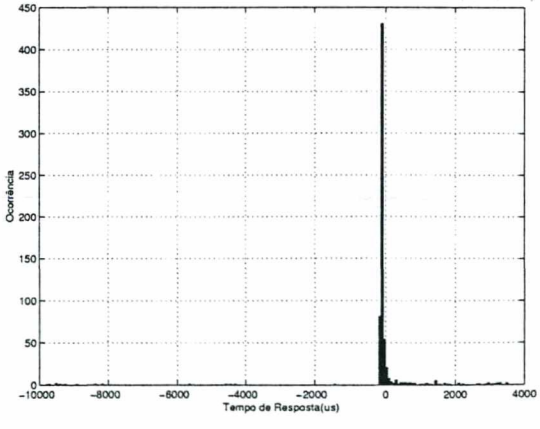


Figura 5.81
Histograma parcial de R -gui-TR9carga T3

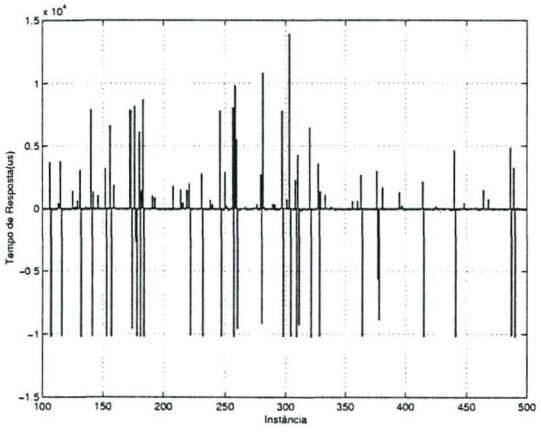


Figura 5.82
Gráfico de R - gui-TR7carga T4

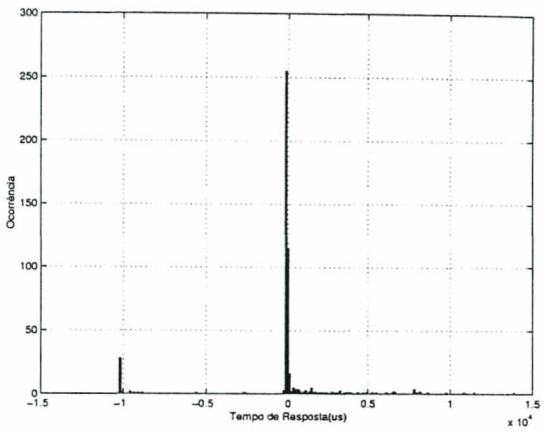


Figura 5.83
Histograma de R -gui-TR7carga T4

O que se verifica pela execução destes experimentos com a aplicação TR7 é o aumento gradativo dos tempos de resposta médio de suas tarefas enquanto o tempo de resposta de pior caso permanece na mesma faixa de valores.

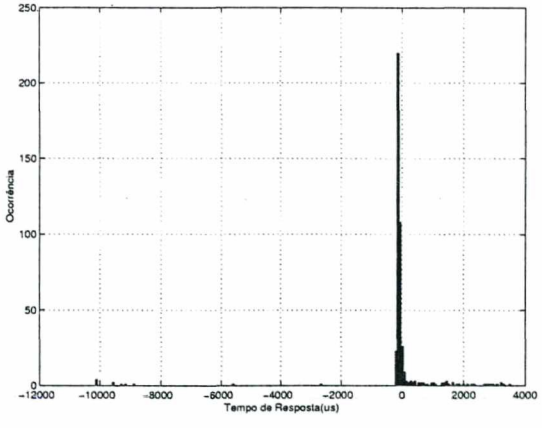


Figura 5.84
Histograma parcial de R - gui-TR7carga T4

Devido a extensão da variação de tempo de resposta das tarefas desta aplicação neste cenário considerando o número total de instâncias, o histograma das tarefas não permite uma boa visualização da distribuição dos tempos de resposta.

O cenário de execução da aplicação TR9 com carga adicional é o cenário com maior quantidade de tarefas presentes no sistema. Como esperado, em vista os resultados obtidos pelos cenários anteriores, TR9 em cenário com carga adicional é a aplicação com maiores

variações de tempo de resposta, bem como valores de desvio padrão dos mesmos de cada tarefa.

Como pode ser visto no gráfico 5.85, o comportamento desta tarefa não apresenta regularidade quanto à curva descritas pelos tempos de resposta. Oscilações bruscas no tempo de resposta de uma instância para a instância seguinte ocorrem durante todo o experimento. A tabela 5.8 ilustra os dados relacionados a uma das sessões de experimento neste cenário.

Tarefa	R				R(90%)			
	Rmin	Rmax	Variação	Desvio padrão	Rmin	Rmax	Variação	Desvio padrão
T0	-17 994	51 137	69 131	10 488.01	-10 070	24 202	34 272	8 709.87
T1	-10 126	47 777	57 903	10 513.46	-10 094	23 291	33 385	8 857.40
T2	-10 146	31 696	41 842	10 062.81	-10 120	22 899	33 019	8 513.65
T3	-10 175	45 732	55 907	10 499.69	-10 138	24 157	34 295	8 671.48
T4	-10 202	47 749	57 951	11 030.26	-10 146	24 674	34 820	9 248.02

Tabela 5.8: gui-TR9carga (Valores em microssegundos)

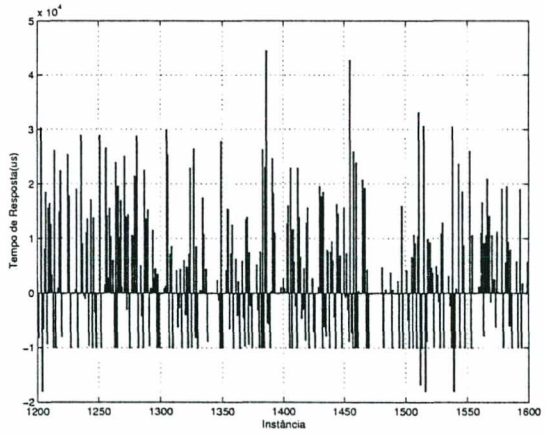


Figura 5.85
Gráfico de R - gui-TR9carga T0

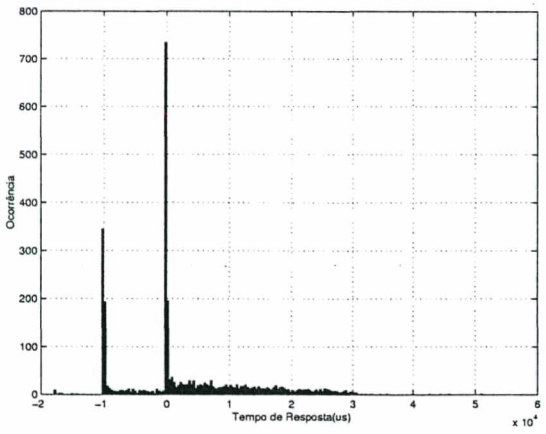


Figura 5.86
Histograma de R -gui-TR9carga T0

O gráfico de histograma das tarefas deste cenário (figuras 5.86 a 5.99) são descrito por dois picos de ocorrências que se concentram sobre valores de tempos distantes aproximadamente de 10ms entre si. As demais ocorrências se distribuem pelo gráfico sendo a maior parte destas sobre tempos de reposta maiores que os tempos correspondentes aos tempos dos picos do histograma.

Poucas instâncias do cenário apresentam tempo de IF maior que alguns microssegundos, entretanto, ocorre em algumas sessões, instâncias com tempo de IF de dezenas de milisse-

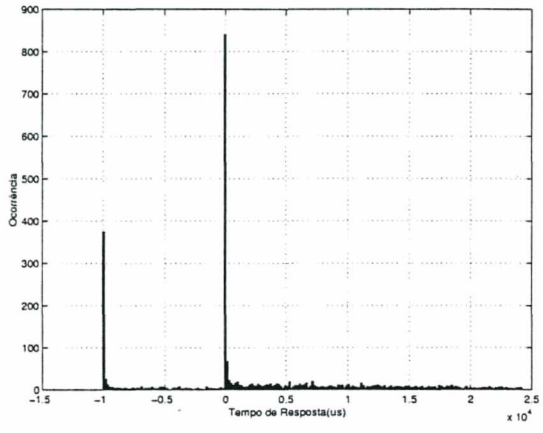


Figura 5.87
Histograma parcial de R - gui-TR9carga T0

gundos (35ms). As instâncias com maior tempo de IF são raras nem sempre correspondem às tarefas com maior tempo de resposta do experimento.

A distribuição de ocorrências de acordo com o histograma, como visto nos demais cenários de TR9, concentra um grande número de instâncias sobre dois valores de tempo de resposta distantes de 10ms.

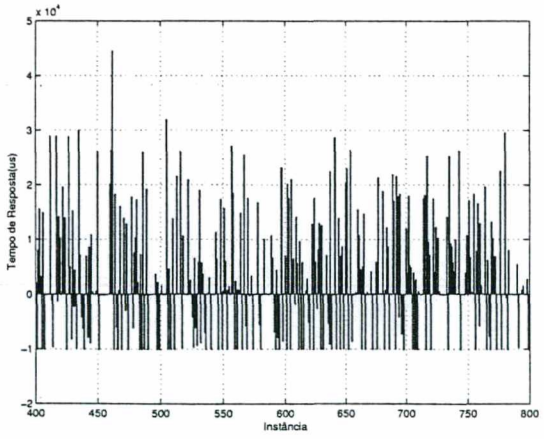


Figura 5.88
Gráfico de R - gui-TR9carga T1

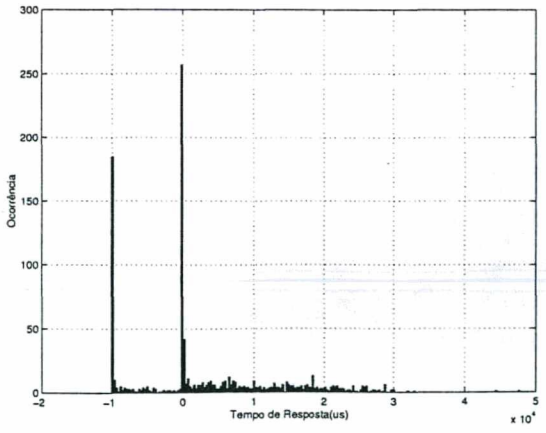


Figura 5.89
Histograma de R -gui-TR9carga T1

Os gráficos de T1 são ilustrados pelas figuras 5.88, 5.89 e 5.87.

Embora existam variações quanto aos valores de tempo de resposta limites, a curva do gráfico e suas características se mantém nos gráficos das outras tarefas. Diferenças entre os

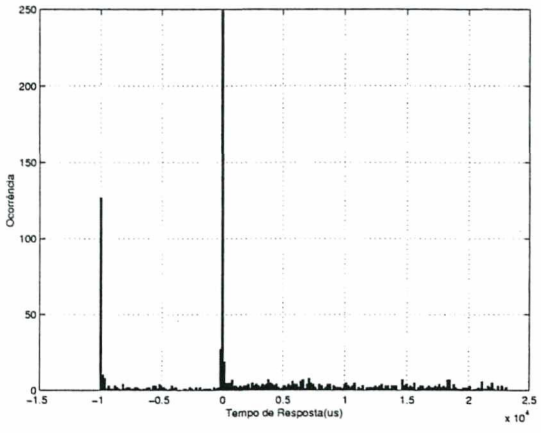


Figura 5.90
Histograma parcial de R -gui-TR9carga T1

valores limites de cada tarefa são mínimos e medidos em microssegundos.

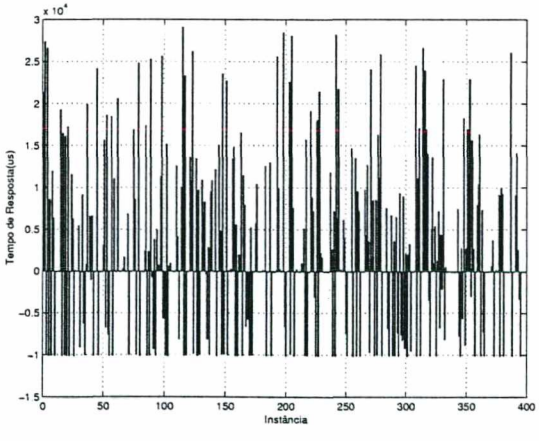


Figura 5.91
Gráfico de R - gui-TR9carga T2

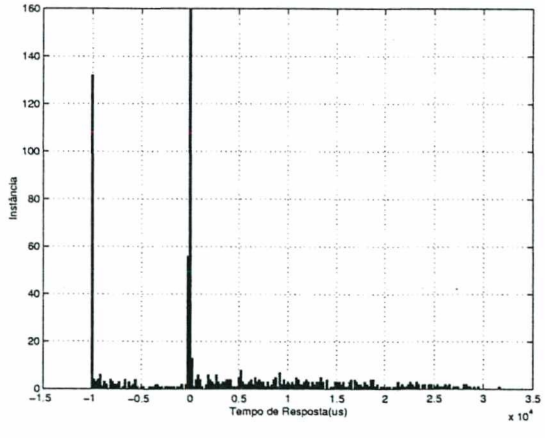


Figura 5.92
Histograma de R -gui-TR9carga T2

Verifica-se neste cenário que existe um equilíbrio entre as tarefas de tempo real, independente de seus períodos (e conseqüente prioridade). As tarefas apresentam variações de tempo de resposta bastante próximos, bem como a variação dos valores de desvio padrão das mesmas. O que também é percebido pela análise do gráfico de seqüência das mesmas, é que embora a tarefa T0 apresente maior prioridade, é a tarefa que apresenta maior variação de tempo de resposta (tarefa com maior freqüência). A variação de tempo de resposta e valores de desvio padrão das tarefas de TR9 em cenário com carga são apresentadas na tabela 5.8.

Tanto através das tabelas 5.4, 5.5 e 5.8, quanto através dos histogramas, observa-se que os

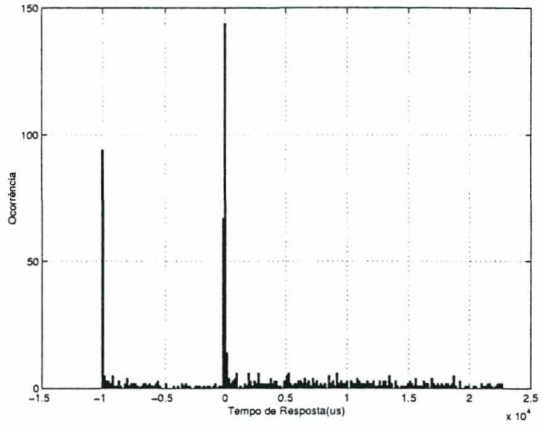


Figura 5.93
Histograma parcial de R -gui-TR9carga T2

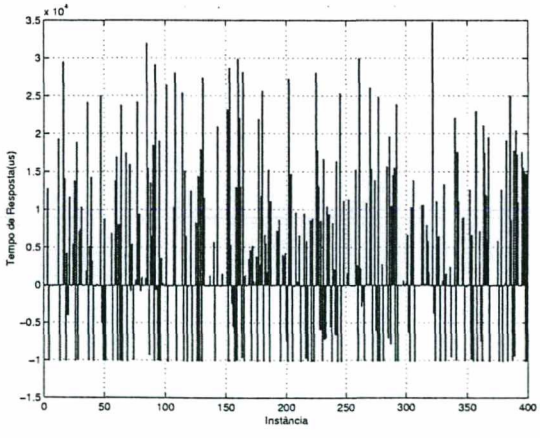


Figura 5.94
Gráfico de R - gui-TR9carga T3

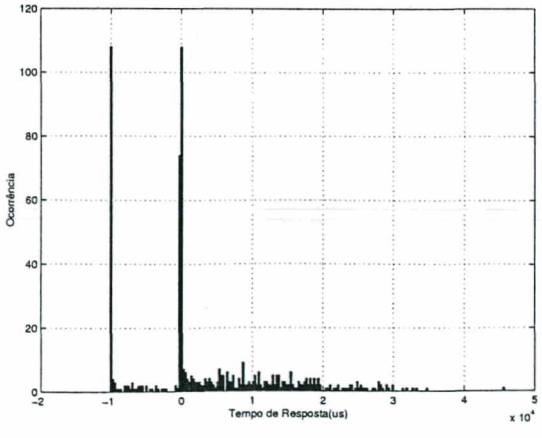


Figura 5.95
Histograma de R -gui-TR9carga T3

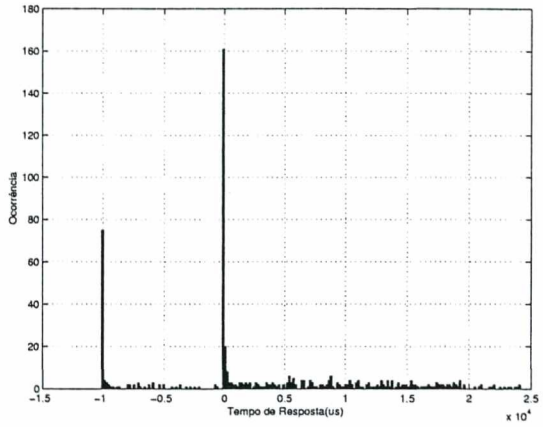


Figura 5.96
Histograma parcial de R -gui-TR9carga T3

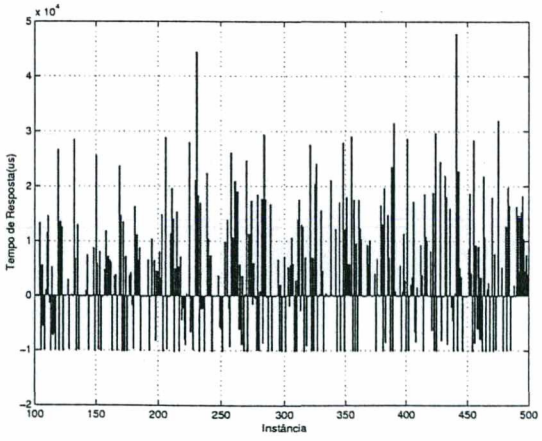


Figura 5.97
Gráfico de R - gui-TR9carga T4

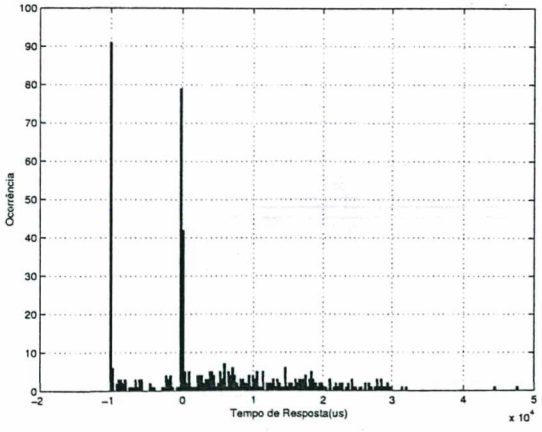


Figura 5.98
Histograma de R -gui-TR9carga T3

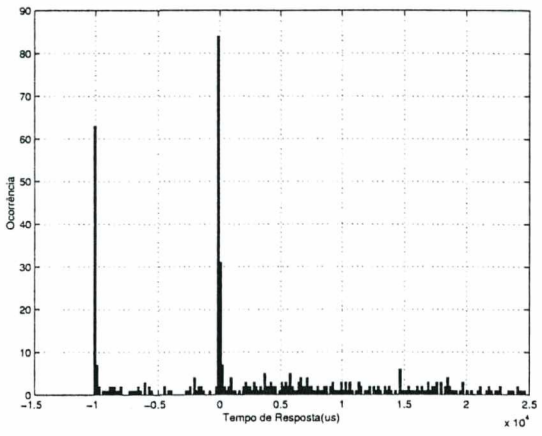


Figura 5.99
Histograma parcial de R -gui-TR9carga T4

resultados obtidos pela execução dos cenários envolvendo a aplicação TR9 são semelhantes. O padrão de comportamento temporal adotado pela aplicação independe de carga adicional. As próprias tarefas não tempo real da aplicação definem este padrão logo nos primeiros cenários descritos para TR9. As modificações sofridas por esta aplicação no decorrer da seqüência de cenários descritos é pequena em relação aos valores apresentados, e percebidas apenas através das tabelas.

Observando o quadro geral de sessões de experimentos dentro de cada cenário, verifica-se que a variação de tempo de resposta obtida nem sempre fornece muitas informações a respeito do comportamento temporal da aplicação. Tarefas podem apresentar variação de tempo de resposta extensa, porém provocadas por uma única ocorrência de R máximo durante a sessão de experimento. A ocorrência de um único tempo de resposta de valor máximo em uma sessão faz com que a variação de R seja ampliada, embora a sessão apresente um comportamento diferente da sessão com vários tempos de R máximo, esta medida não poderá indicar tais diferenças, somente uma visão parcial sobre os dados.

Pelo mesmo motivo, considerar somente o valor de desvio padrão não garante a análise correta sobre o comportamento destas tarefas no tempo. Em grande parte dos casos, o desvio padrão permite que se tenha uma idéia da distribuição de tempos de resposta dentro da variação de R de cada experimento. Ainda assim, esse valor é bastante influenciado por ocorrências isoladas de pior tempo na sessão. Por outro lado, a variação dos valores de desvio padrão mostram que dentro de faixas de valores semelhantes, a variação entre os tempos de resposta de cada instância têm valores bastante diferentes.

A verificação dos valores obtidos a partir da análise de 90% de instâncias também per-

mite que se obtenha melhor conhecimento sobre cada experimento, bem como a construção de gráficos mais informativos. Ainda assim, somente o conjunto de todos estes elementos permitem a análise mais completa sobre os dados obtidos em cada sessão.

Quanto às restrições temporais de cada tarefa, mesmo em cenário com carga, a maior parte das instâncias das tarefas de tempo real conseguem manter deadline igual ao período, tendo em vista a variação de tempo de resposta dentro do qual elas concluem. Embora exista a ocorrência de intervalos de tempo entre o início e o fim da execução da tarefa maiores (valores em milissegundos), mais de 98% das instâncias executadas apresentam IF em poucos microssegundos. Os intervalos de IF de milissegundos são relativamente raros dentro de cada experimento, e passam a existir a partir da execução dos cenários com 9 tarefas. Até então, interferências ocorridas entre tarefas eram capturadas somente através dos tempos de R. Ainda assim, não existe sempre a relação direta entre os maiores tempos de IF e maiores tempos de R nos cenários executados.

5.5 Execução em máquinas diferentes

Embora as aplicações e cenários ilustrem aspectos interessantes quanto ao comportamento de tarefas de tempo real no Linux padrão, não se pode concluir até que ponto a interferência percebida pelos resultados descritos se devem unicamente ao sistema operacional e ao cenário em si. Considerações quanto ao efeito do perfil de hardware em que executam os experimentos são perdidos nesta análise.

Esta seção descreve as conclusões obtidas a partir da execução de sessões de experimentos realizados em outro hardware. Esta segunda máquina, Máquina2 (a máquina até então adotada será referenciada por Máquina1), utiliza o mesmo kernel da Máquina1. Tanto as aplicações de tempo real quanto a interface gráfica de usuário são também idênticas em ambas as máquinas.

A Máquina1, em que foram executadas todos os cenários descritos na seção anterior, é um AMDK6 500MHz com 128MB de memória RAM. A Máquina2 é um Pentium 200MHz com 64MB de memória RAM.

O objetivo desta seção é comparar dados das aplicações de tempo real em máquinas diferentes a fim de observar se existem diferenças no padrão de comportamento apresentado pelas tarefas de tempo real no Linux dependentes do tipo de hardware adotado em sua execução.

Embora sessões de experimento de todos os cenários descritos neste capítulo tenham sido executadas em ambas as máquinas, foram escolhidos somente três cenários para ilustração

dos resultados da comparação. Os cenários ilustram o comportamento das três aplicações de tempo real construídas, executando no ambiente com interface gráfica e sem carga adicional; gui-TR1, gui-TR7 e gui-TR9.

De modo geral, o comportamento da tarefa de tempo real em ambas as máquinas são muito parecidos. Nos cenários que utilizam as aplicações TR1 e TR7 foi observado um pequeno aumento na faixa de variação, e valor de desvio padrão calculado para a mesma. Este aumento foi percebido executando diversas sessões de experimento em ambas as máquinas e contabilizando o resultado geral, não se trata de valores obtidos na Máquina2 que nunca são apresentados pela Máquina1, mas apenas uma alteração no comportamento médio.

Tanto a aplicação TR1 quanto TR7 nos cenários sem carga apresentam tempos de resposta em microssegundos. Sendo esta grandeza pouco significativa no contexto do Linux padrão, comparações em relação ao comportamento das tarefas em máquinas diferentes se baseiam principalmente nos gráficos de distribuição de ocorrências, o histograma de tempo de resposta das tarefas.

Os gráficos 5.100 e 5.101 ilustram o histograma da tarefa da aplicação TR1 em cada máquina. A curva do histograma de ambas as tarefas se mostram bastante semelhantes, mas com um acúmulo maior de ocorrências na cauda do histograma da Máquina2. Embora o acúmulo indique um maior número de ocorrências com tempo de resposta mais alto, trata-se de um número reduzido de ocorrências. Os gráficos 5.102 e 5.103 ilustram o histograma parcial da mesma tarefa.

Ao contrário das seções anteriores, os gráficos apresentados nesta seção possuem suas escalas ajustadas para permitir melhor comparação entre histogramas de tarefas em máquinas diferentes. Assim, mesmo existindo um limite máximo e mínimo distantes do acúmulo visível de ocorrências do histograma, não implica na existência de alguma instância próxima destes valores.

O histograma parcial da tarefa construído a partir de 90% das instâncias da mesma, fornece uma visão mais detalhada da curva de distribuição de ocorrências nesta tarefa. Apesar de apresentar distribuição dentro de valores limite diferentes, os gráficos ilustram um tipo de curva semelhante, com o maior número de ocorrência em um tempo de R mais próximo do limite inferior.

Em cenários com maior número de tarefas, entretanto, o que foi percebido pela execução das sessões de experimento é que em geral, na Máquina2 os resultados apontam para uma faixa de variação e desvio padrão de R pouco maior que na Máquina1. De fato, a realização dos experimentos em cenários idênticos na Máquina2 resulta em gráficos de histograma que apresentam, como característica geral, maior número de ocorrências de instâncias na cauda

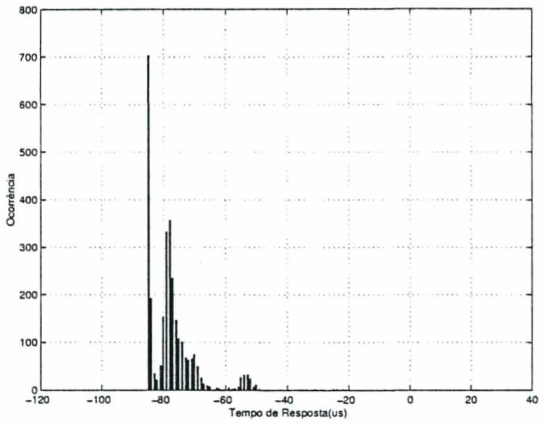


Figura 5.100
Histograma de R - gui-TR1 Máquina1

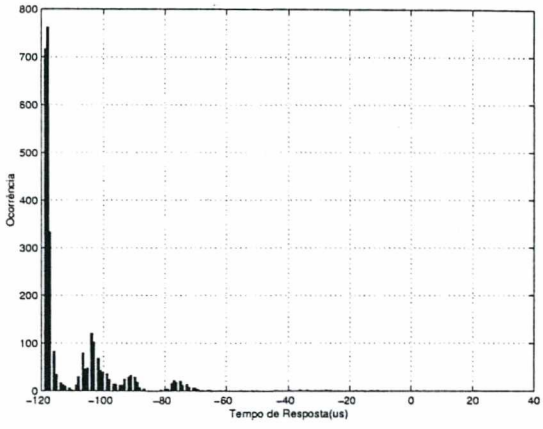


Figura 5.101
Histograma de R - gui-TR1 Máquina2

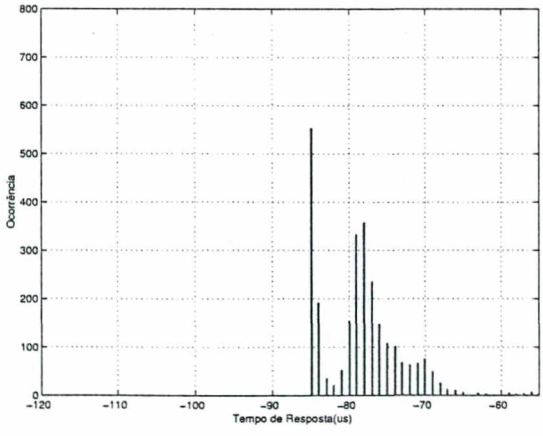


Figura 5.102
Histograma parcial de R - gui-TR1
Máquina1

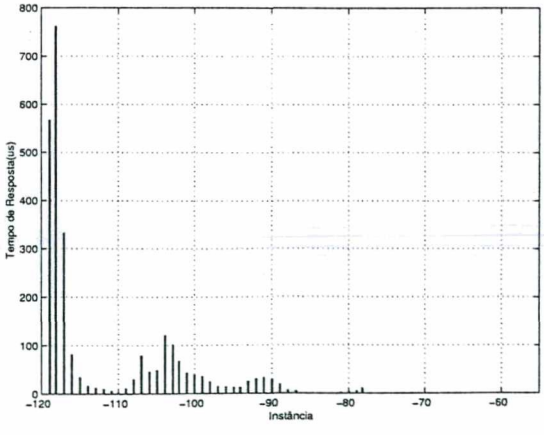


Figura 5.103
Histograma parcial de R - gui-TR1
Máquina2

do gráfico. Ajustando os gráficos de histograma para a mesma escala de valores (estabelecendo limites iguais para ambos os gráficos), obtidos nas duas máquinas, verifica-se que a curva de distribuição da Máquina2, apesar de apresentar o formato similar, é bem mais dispersa no domínio do tempo.

Os gráficos 5.104 e 5.105 ilustram respectivamente, o histograma da tarefa T0 do cenário gui-TR7 em um experimento na Máquina1 e na Máquina2. Histogramas da tarefa T1 destes mesmos experimentos, são ilustrados nas figuras 5.106 e 5.107.

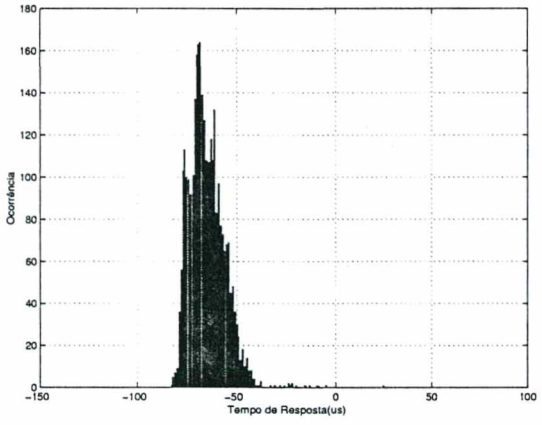


Figura 5.104
Histograma de R - gui-TR7 T0 Máquina1

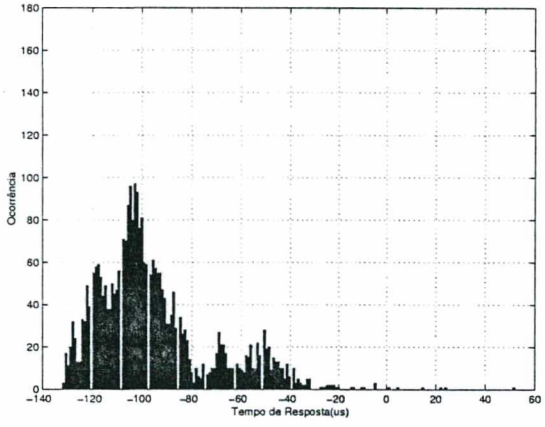


Figura 5.105
Histograma de R - gui-TR7 T0 Máquina2

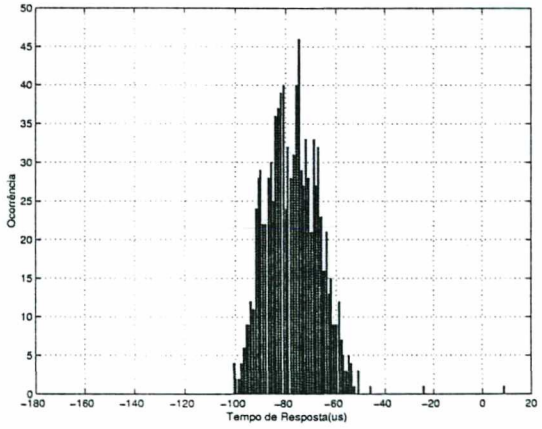


Figura 5.106
Histograma de R - gui-TR7 T1 Máquina1

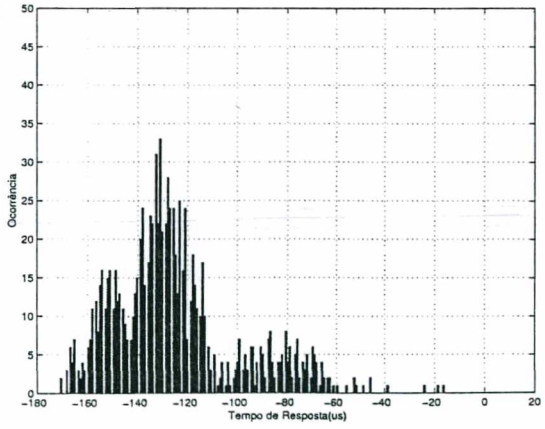


Figura 5.107
Histograma de R - gui-TR7 T1 Máquina2

Assim como descrito anteriormente, os histogramas das tarefas da aplicação TR9 ilustrados nos gráficos 5.108, 5.109, 5.110 e 5.111 apresentam o mesmo tipo de curva de

distribuição nas duas máquinas.

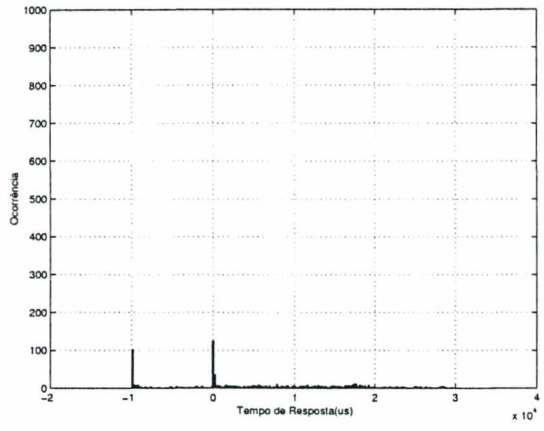


Figura 5.108
Histograma de R - gui-TR9 T0 Máquina1

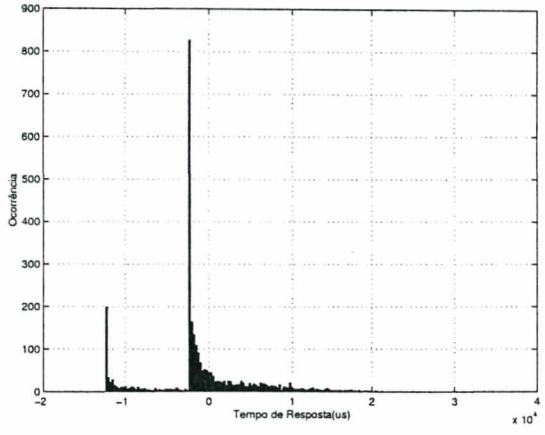


Figura 5.109
Histograma de R - gui-TR9 T0 Máquina2

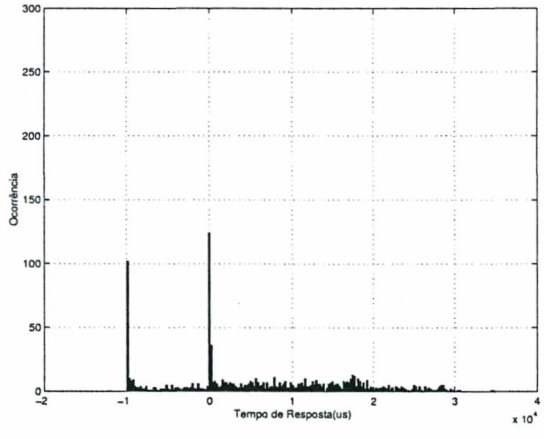


Figura 5.110
Histograma de R - gui-TR9 T1 Máquina1

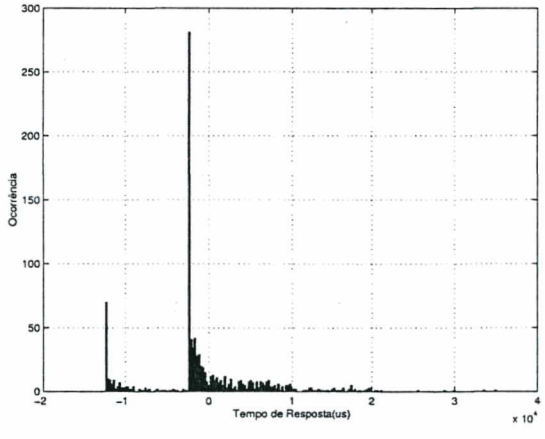


Figura 5.111
Histograma de R - gui-TR9 T1 Máquina2

Diferente das aplicações TR7 e TR1, a aplicação TR9 apresenta tempo de resposta de suas tarefas medidos em milissegundos. As tarefas de acesso a disco e rede garantem a esta aplicação um cenário mais complexo que as aplicações anteriores. As requisições de serviços destes dispositivos provocam chamadas a processos do kernel com maior freqüência, provocando a execução de trechos mais extensos do kernel com interrupções desabilitadas e conseqüentemente provocando tempos maiores de interferência nas tarefas de tempo real.

5.6 Conclusão

Observando o quadro geral de sessões de experimentos dentro de cada cenário, verifica-se que a faixa de variação obtida nem sempre fornece muitas informações a respeito do comportamento temporal da aplicação. Tarefas podem apresentar faixas de variação extensas porém provocadas por uma única ocorrência de pior caso durante a sessão de experimento. A ocorrência de um único tempo de resposta de pior caso em uma sessão faz com que sua faixa de valores seja ampliada, e embora a sessão apresente um comportamento diferente de uma sessão com vários tempos de pior caso, esta medida não poderá indicar tais diferenças, somente uma visão parcial sobre os dados.

Pelo mesmo motivo, considerar somente o valor de desvio padrão não garante a análise correta sobre o comportamento destas tarefas no tempo. Em grande parte dos casos, o desvio padrão permite que se tenha uma idéia da distribuição de tempos de resposta dentro da faixa de variação de cada experimento. Ainda assim, esse valor também é bastante influenciado por ocorrências isoladas de pior tempo de resposta, por exemplo. Por outro lado, diferentes valores de desvio padrão dentro de faixas de valores semelhantes, ilustram a variação entre os tempos de resposta de cada instância dentro de uma sessão de experimento.

A verificação dos valores obtidos a partir da análise de 90% de instâncias também permite que se obtenha melhor conhecimento sobre o comportamento típico de cada experimento, bem como a construção de gráficos mais informativos. Ainda assim, somente o conjunto de todos estes elementos permite a análise mais completa sobre os dados obtidos em cada sessão.

Considerando gráficos e tabelas, o que pode ser verificado é que o Linux padrão oferece um bom comportamento temporal quando executando aplicações simples de tempo real. Obviamente é inviável utilizá-lo para aplicações de tempo real críticas, mas existe uma quantidade razoável de aplicações de tempo real soft compatíveis com seu comportamento.

As aplicações TR1 e TR7 provêem tempo de resposta com variações de microssegundos. Embora a função *sleep*, forneça um bom desempenho na maior parte das ativações das tarefas, eventuais interferências provocadas por atividades internas ao kernel, aliadas ao efeito da resolução do timer do Linux (inapropriada para tempo real), provocam tempo de resposta bem maior que o tempo de resposta típico, para algumas instâncias da tarefa.

A aplicação TR1 é a aplicação mais simples utilizada nos experimentos e apresenta faixa de variação de tempo de resposta em dezenas de microssegundos. O histograma construído pela listagem de seus tempos de resposta descreve uma curva com uma cauda longa, porém que apresenta pequena quantidade de ocorrências. Esta característica favorece aplicações

que admitem uma certa margem de tolerância em relação aos piores tempos de resposta obtidos durante a execução. Quando considerado somente 90% das ativações desta tarefa, todas as perturbações ocorridas durante sua execução são excluídas do histograma.

No cenário mais simples, texto-TR1, em que somente processos do sistema operacional podem provocar interferência na tarefa de tempo real, as ocorrências mais freqüentes durante a sessão de experimentos são de instâncias com tempo de resposta igual ao limite mínimo observado de R da tarefa. À medida que os cenários se tornam mais complexos, os histogramas tendem a apresentar maior dispersão das ocorrências.

A inclusão de interface gráfica de usuário no ambiente de execução das tarefas de tempo real provocam, quando perceptível, um aumento muito pequeno nos valores usados para a análise de resultados. Isto pode ser observado nos cenários das três aplicações de tempo real descritas. No cenário da aplicação TR1, a inclusão de interface gráfica provoca o aumento da dispersão de ocorrências sobre o domínio de tempo de resposta. Embora tal fato implique na existência de interferência por parte da interface gráfica de usuário, uma tarefa não tempo real, o aumento da dispersão é pequeno, e a variação do R da tarefa permanece menor que 100us (exceto nos casos da ocorrência de erro de *sleep*, que gera tempo de resposta de até 10ms).

Os resultados em números e em gráficos da execução das demais aplicações não são suficientemente distintos a ponto de indicar a presença ou não de interface gráfica de usuário no momento da execução do experimento. Observando os resultados dos experimentos nos cenários sem carga com as aplicações TR1 e TR7 verifica-se que o efeito provocado pela inclusão de interface gráfica de usuário no ambiente de execução da aplicação de tempo real é maior em aplicações mais simples. Em aplicações com várias tarefas, a interferência provocada pela interface gráfica de usuário tende a se tornar menos visível face às interferências provocadas pelas próprias tarefas de tempo real da aplicação.

A aplicação TR7 nos cenários sem carga, apresenta um aumento gradativo de variação e desvio padrão de R de suas tarefas a medida que o período destas aumenta. O efeito visível deste aumento nos histogramas das tarefas é o aumento da dispersão das ocorrências sobre o domínio de tempo de resposta da mesma. Os resultados obtidos com cenários da aplicação TR7 ilustram também que tarefas de maior prioridade são favorecidas dentro da aplicação.

Ainda que contenha tarefas não tempo real como parte da aplicação, TR7 apresenta tempo de resposta em microssegundos, o que permite supor que tarefas simples não tempo real não provocam grande impacto sobre a execução de tarefas de tempo real.

A análise de resultados obtidos a partir dos cenários com a aplicação TR9, mostram que as aplicações que envolvem tarefas não tempo real com uso intenso de dispositivos de rede

e disco provocam muito mais interferências que aquelas de teclado e vídeo. O tempo de resposta das tarefas desta aplicação são medidos em milissegundos, três ordens de grandeza maior que os tempos medidos pela aplicação TR7. A execução das tarefas não tempo real que fazem o uso intensivo de dispositivos de rede e disco, provoca um grande número de chamadas a funções do kernel do Linux. Uma vez que se trata de um kernel não preemptivo com várias regiões de interrupções desabilitadas, a interferência provocada pelo próprio sistema operacional sobre tarefas de tempo real são bastante grandes.

Assim como observado nas seções de descrição dos cenários com as demais aplicações, TR9 não apresenta diferenças grandes de valores de tempo quando adicionado ao seu ambiente de execução, a interface gráfica de usuário. Os tempos indicados na tabela e os gráficos de histograma permanecem idênticos em texto-TR9 e gui-TR9.

Porém, o aumento gradativo de valores, percebido na execução de TR7 não ocorre com TR9. Os valores gerados pela execução de cada tarefa desta aplicação e seus gráficos de histograma correspondentes são idênticos em ambos os cenários sem carga. Ao considerar somente os 90% de instâncias com tempo de resposta mais próximos do tempo de resposta médio de cada tarefa, o valor da variação das tarefas são próximos entre si. E embora se trate de cinco tarefas com cinco prioridades diferentes, todas as tarefas apresentam gráfico de seqüência e histograma similar. O histograma apresentado pelos cenários compostos por TR9, descreve uma curva de distribuição dividida em dois picos de ocorrências de maior concentração de instâncias. Tais valores aparecem distantes de 10ms entre si, porém cada um deles com uma longa cauda de ocorrências de instâncias com tempo de resposta maiores.

Os cenários com carga, apresentam tempo de resposta em milissegundos para todas as aplicações. Mesmo a aplicação mais simples, TR1, sofre um grande impacto em seu comportamento temporal diante da presença da carga adicional e apresenta variação e desvio padrão em milissegundos.

Nos cenários com carga, o comportamento da tarefa de 50ms em TR1 e em TR7 são muito parecidos, tanto quanto aos valores indicados na tabela, quanto à aqueles visíveis pelo histograma destas tarefas. A interferência provocada pela presença da carga adicional no ambiente de execução destas tarefas encobre as pequenas interferências que caracterizavam as diferenças de comportamento destas tarefas nos cenários sem carga.

Pelo mesmo motivo, as diferenças entre as tarefas de TR7 passam a ser bem menos perceptíveis em gui-TR7carga. Assim como foi observado com TR9, TR7 passa a apresentar tarefas com comportamento similar entre si no cenário com carga. A interferência provocada entre tarefas de tempo real é pequena em comparação com a interferência provocada pela carga adicional, e embora esteja presente no tempo de resposta das tarefas de TR7, passam

a ser menos evidentes. Embora a variação dos tempos de suas tarefas não ultrapasse 50ms (menor período da aplicação), a tarefa de maior prioridade não é a tarefa de menor variação.

A atribuição de prioridades da classe de tempo real para as tarefas da aplicação estabelece uma ordem de preferência entre elas que é respeitada pelo escalonador. Entretanto, as maiores interferências sofridas pelas tarefas de tempo real são provocadas pelas tarefas convencionais e por atividades dentro do kernel. Tais interferências atingem igualmente todas as tarefas de tempo real sem distinção de prioridades provocando o resultado observado nos experimentos com TR7 e TR9.

A aplicação TR9, diante do cenário com carga, não apresentou grande modificação quanto ao tempo de resposta de suas tarefas. O efeito da carga para esta aplicação é análogo ao efeito de interface gráfica nos cenários texto-TR9 e gui-TR9. Existe, em geral, um pequeno aumento nos números de TR9 no cenário com carga, porém não o suficiente para que se possa distinguir experimentos dos dois cenários através de seus resultados. As tarefas desta aplicação permanecem idênticas uma às outras pois a interferência provocada pela carga adicional está contida em meio a interferência provocada pelas tarefas de acesso a rede e disco.

O fato da tarefa de maior prioridade, tanto em TR7carga, quanto nos cenários da aplicação TR9, apresentar maior variação, é conseqüência direta do valor do período da tarefa. Num mesmo intervalo de tempo, a tarefa de maior prioridade possui mais ativações que as demais tarefas da aplicação, permanecendo no sistema por mais tempo e portanto, sujeita a mais interferências.

Quanto às restrições temporais de cada tarefa, mesmo em cenário com carga, a maior parte das instâncias das tarefas de tempo real consegue manter o deadline igual ao período, tendo em vista a variação de tempo dentro do qual elas concluem. Embora exista a ocorrência de intervalos de tempo entre o início e o fim da execução da tarefa maiores (valores em milissegundos), mais de 98% das instâncias executadas apresentam IF em poucos microssegundos. Os intervalos de IF de milissegundos são relativamente raros dentro de cada experimento, e passam a existir a partir da execução dos cenários com 9 tarefas. Até então, interferências ocorridas nas tarefas eram capturadas somente através dos tempos de R. Ainda assim, não existe sempre a relação direta entre os maiores tempos de IF e maiores tempos de R nos cenários executados.

Os experimentos em uma segunda máquina ilustra apenas que embora os números fornecidos sejam diferentes, em geral mais altos devido a capacidade da máquina em questão, o comportamento geral da tarefa se mostra o mesmo. As mesmas deficiências do Linux apontadas pelas tarefas de tempo real na Máquina1 estão presentes também na Máquina2.

O que pode ser dito tendo em vista os resultados obtidos pelos experimentos descritos neste capítulo é que para aplicações simples, o aumento na resolução do timer do Linux seria uma grande contribuição para melhora de desempenho de tarefas de tempo real. A diminuição dos atrasos decorrentes de erros no*sleep* destas tarefas tornaria o Linux adequado para um número muito maior de aplicações.

Entretanto, a resolução do timer do Linux não é o único fator a contribuir para a deficiência em seu desempenho quando se trata de aplicações de tempo real. Uma vez que este sistema é projetado para aplicações convencionais, a política de distribuição de recursos leva em consideração a justiça no atendimento de requisições, e permite que tarefas comuns sejam atendidas antes da conclusão de tarefas de tempo real. Desta maneira, existe a ocorrência de inversão de prioridade, e embora tarefas de tempo real tenham prioridade maior que tarefas comuns, devem esperar até que tarefas comuns tenham suas requisições atendidas pelo kernel.

Sendo o kernel do Linux, não preemptivo, a presença de tarefas mais prioritárias na fila de prontos não provoca a troca de contexto de tarefas. Tarefas de tempo real, mesmo prontas para executar, devem esperar a conclusão de atividades disparadas dentro do kernel até que seja encontrado neste, um ponto de preempção. Esta característica do sistema se torna evidentemente prejudicial a tarefas de natureza temporal quando tarefas convencionais de uso intenso de dispositivos de rede e disco estão presentes no ambiente de execução.

Considerando o que foi observado durante os experimentos, verifica-se que os atrasos em ativações das tarefas de tempo real podem ser provocadas por uma série de fatores decorrentes de aspectos inerentes ao funcionamento do próprio Linux. O fato de adotar uma resolução de interrupções de timer de 10ms, proporciona uma margem de atraso muito grande para tarefas de tempo real. Embora as chamadas a função de *sleep* sejam feitas baseadas no instante corrente, não há garantias de que esta chamada será processada neste instante, ou mesmo num instante próximo. O máximo que se pode afirmar é que esta chamada será processada no instante do *tick* de *clock* mais próximo do instante em que foi feita a chamada. Ainda assim, isso deve ocorrer caso nenhuma atividade dentro do kernel atrapalhe o atendimento desta função, caso não existam tarefas que possam interferir no processamento desta chamada. Mesmo executando sem qualquer outra aplicação no sistema, tarefas auxiliares do kernel são processadas periodicamente, fazendo a manutenção do sistema. Assim como as interferências provocadas por outras tarefas de tempo real no sistema, a interferência provocada por este tipo de tarefa é contabilizada tanto nos tempos de resposta quanto nos tempos de IF de cada instância.

No caso do sistema estar ocupado com tarefas mais prioritárias no momento em que a chamada de *sleep* foi feita, a chamada deve esperar, e com a espera existe a desatualização do

intervalo de tempo usado como parâmetro da chamada *sleep*. Vários *ticks* de *clock* podem ser perdidos nesta esta situação, de fato, não existe um limite para este tempo de atraso, somente dados estatísticos que baseiam a probabilidade desse tempo. Situação semelhante é encontrada novamente pela tarefa de tempo real no momento em que a tarefa cumpre seu tempo em *sleep* e deve retornar a fila de prontos. Retornar a fila de prontos é uma nova requisição da tarefa que será atendida dentro das mesmas condições de atendimento da chamada de *sleep*. O instante em que é terminado o prazo de *sleep* da tarefa pode não ser o instante em que esta será colocada na fila de prontos. Toda interferência presente no sistema durante estas atividades é contabilizada no tempo de resposta da instância em questão e caracteriza o *jitter* de liberação da ativação.

Tendo em vista a magnitude da interferência provocada no cenários da aplicação TR9 em comparação com os cenários de TR7, verifica-se que as tarefas de acesso a rede e disco provocam a execução de longos trechos de interrupção desabilitada dentro do kernel, resultando em tempos de resposta muito maiores por parte de tarefas de tempo real desta aplicação. Este tipo de interferência sofrido pela tarefa de tempo real é contabilizada tanto em tempos de resposta das ativações quanto em tempos medidos entre o início e o fim da execução da tarefa(IF). Para este tipo de problema apresentado por sistemas de propósito geral, a troca da política de escalonamento aliada ao aumento de pontos de preempção no kernel poderiam significar o aumento significativo de aplicabilidade do Linux padrão para sistemas de tempo real.

A realização dos experimentos permitiu verificar não apenas uma série de aspectos relacionados ao comportamento de tarefas de tempo real do Linux, mas também aspectos práticos relacionados a implementação de tarefas de tempo real de características diversas. Os resultados obtidos indicam um comportamento que nem sempre pode ser previsto por modelos matemáticos ou teorias relacionadas a tarefas de tempo real. Mais influente que os elementos descritos por estes modelos e teorias, é o impacto provocado pelas características de implementação das tarefas e o funcionamento do sistema em que executam, que acabam por ditar o comportamento das mesmas.

Capítulo 6

Conclusões

A implementação de aplicações de tempo real totalmente previsíveis é uma tarefa difícil. A passagem do projeto de uma aplicação de tempo real partindo de um modelo, para a sua realização através da concepção num sistema computacional implica na inserção de indeterminismo inerente à linguagens de programação e plataformas de desenvolvimento. A construção de aplicações de tempo real parte da busca pelo equilíbrio entre as propriedades que se deseja da aplicação e as limitações impostas pelas características do sistema operacional e aos meios de construção disponíveis.

A popularização do Linux como plataforma para aplicações diversas é ao mesmo tempo causa e conseqüência de sua constante evolução. A flexibilidade e disponibilidade deste sistema tem atraído a atenção de projetistas de todas as áreas, e sua utilização como plataforma para aplicações de tempo real brandas torna-se cada vez mais comum.

Esta dissertação faz a documentação da construção e execução de aplicações hipotéticas de tempo real sobre o Linux padrão. O objetivo destas atividades foi a observação do comportamento temporal destas aplicações diante de diferentes ambientes de execução ao qual foram expostas.

A observação e análise do comportamento se deu através de tempos de resposta capturados durante a execução das mesmas. Diferentes aplicações e ambientes de execução foram construídos afim de prover variedade de situações e conseqüentemente extrair diversidade de informações.

As informações obtidas a partir dos experimentos fornecem indícios sobre o comportamento temporal do sistema operacional Linux e pode ajudar desenvolvedores na decisão sobre a aplicabilidade do sistema às necessidades específicas de suas aplicações.

Os principais conceitos relacionados a sistemas de tempo real são apresentados no capítulo 2. Este capítulo traz a definição de sistemas de tempo real, a classificação empregada para os mesmos, a descrição do modelo de tarefas periódicas e tópicos sobre o escalonamento de tarefas de tempo real. É ilustrado brevemente o modelo de escalonamento Taxa Monotônica em que se baseiam as tarefas implementadas.

O capítulo 3 mostra os principais aspectos do Linux que estão relacionados ao trabalho realizado. É descrito neste capítulo o funcionamento de mecanismos e características gerais do comportamento do kernel que provocam maior impacto sobre aplicações de tempo real.

O capítulo 4 traz a documentação de todo o processo de construção das aplicações. Foram levantados neste capítulo, os mecanismos disponíveis no Linux através dos quais é possível implementar tarefas periódicas. Uma vez que não existem APIs voltadas para esta finalidade no Linux padrão, a escolha do mecanismo foi feita buscando aproximar o comportamento temporal das mesmas do comportamento ideal. As limitações impostas por esta adaptação e as razões que levaram à escolha do mecanismo são também expostas no capítulo.

Experimentos preliminares de medições e comparações mostraram que existem métodos simples de ajuste das tarefas. Ajustes implementados através de valores adicionados ao cálculo de parâmetros de chamadas de atraso, ou do uso das mesmas chamadas objetivando sincronização de grades de tempo permitem a melhoria do comportamento temporal da tarefa de tempo real.

Entretanto, propriedades tais como a inadequação da granularidade do temporizador do Linux para tarefas de tempo real e o fato do kernel ser não preemptivo não podem ser superadas pela adoção de mecanismos simples.

No capítulo 5, são apresentados os resultados obtidos a partir da execução das aplicações implementadas. Histogramas mostram que a distribuição das ocorrências em cenários simples se restringem a uma faixa de tempos de resposta bastante curtos, medidos em dezenas de microssegundos. Embora as funções de inativação forneçam um bom desempenho na maior parte das ativações das tarefas, eventuais interferências provocadas por atividades internas ao kernel, aliadas ao efeito da resolução do temporizador do Linux (inapropriada para tempo real), provocam atrasos bem maiores que o tempo de resposta típico, para algumas instâncias da tarefa.

Em cenários mais complexos e nos cenários com carga adicional no sistema, estes atrasos se tornam mais freqüentes, tendo em vista a maior quantidade de interferência provocada por atividades de uso intenso de dispositivos como os de rede e disco. Tais cenários proporcionam histogramas de tempos de resposta medidos em milissegundos, com maiores concentrações de valores sobre instantes de tempo distantes de 10ms.

O Linux, apesar de não ter preocupações temporais em seu projeto, permite o bom comportamento temporal de tarefas de tempo real brandas. Mesmo com a adição de tarefas não tempo real simples, os tempos de resposta resultantes podem ainda ser medidos em microssegundos. Porém, a inserção de tarefas de uso intenso de dispositivos tais como disco e rede provocam a degradação considerável de seu comportamento. Além dos fatores já mencionados, o fato do kernel ser não preemptivo e apresentar regiões de interrupção desabilitadas contribuem para o aumento de interferências sobre o tempo de resposta das tarefas.

A realização do trabalho comprovou também deficiências do Linux em disponibilizar facilidades para implementação destas tarefas tais como definidas no modelo de tarefas periódicas. Nem todos os elementos do modelo podem ser identificados nas aplicações e algumas das propriedades apresentadas por elas não são capturadas pelo modelo.

Mesmo sendo possível detectar os principais focos de problemas no Linux com relação ao seu comportamento diante de aplicações de tempo real, verifica-se também que não podem ser facilmente superados. Modificações simples ou isoladas não são capazes de melhorar o comportamento temporal deste sistema para qualquer tipo de aplicação em execução no sistema. A utilização de prioridades e políticas de escalonamento estabelece uma ordem de preferência que de fato é obedecida pelo escalonador. Porém a maior causa de interferência sobre tarefas de tempo real provocadas por atividades dentro do kernel, com origem em tarefas convencionais, que atingem todas as tarefas de tempo real sem distinção de prioridades.

Em decorrência da complexidade do kernel do Linux e da inexistência de maiores facilidades de mecanismos de implementação de tarefas de natureza temporal, informações tais como as apresentadas neste trabalho são obtidas somente através de experimentos. Mesmo o mapeamento do kernel deste sistema, embora pudesse trazer informações mais precisas sobre seu comportamento temporal, não pode ser feito sem agregar em seu conteúdo o indeterminismo próprios de um sistema de propósitos gerais otimizado para o melhor desempenho no caso médio.

As atividades envolvidas neste trabalho envolveram a construção de dezenas de tarefas de tempo real utilizando diferentes estruturas e mecanismos e a execução de várias sessões de experimentos preliminares na busca das aplicações e cenários apresentados nesta dissertação. Cerca de 700 arquivos de dados (pelo menos) foram gerados pelas aplicações e utilizados para basear as informações apresentadas no capítulo 5, que faz o uso de uma amostra na construção de gráficos e tabelas. Dentre os experimentos, estão sessões de duração de alguns minutos até sessões de dez horas de duração.

A maior dificuldade encontrada na etapa de experimentos constitui a verificação de dados resultantes considerando a quantidade de medições que caracterizam uma única sessão. Uma vez que a listagem dos resultados (formato original de saída das aplicações), não permite

clareza suficiente para análise, esta deve ser feita pela construção de gráficos e histogramas. A geração destes, por sua vez, exigiram cuidados quanto a escolha de escalas e unidades sob pena de sacrificar a transparência ou a exatidão dos dados ilustrados.

Existem atualmente, inúmeros projetos de modificações sobre o Linux para o atendimento satisfatório de uma classe maior de sistemas de tempo real. Ainda assim, não há um consenso sobre a supremacia de nenhuma destas abordagens em decorrência da variedade de tipos de tarefas que compõem o possível conjunto a ser executado no sistema.

Diversas questões permanecem em aberto e podem ser objeto de futuros estudos;

- Medições utilizando maior freqüência de temporizador do sistema; Embora esta medida tenha repercussões sobre o desempenho, considerando o aumento do número de interrupções de tempo a serem tratados, não são facilmente encontradas documentações a respeito de um valor de equilíbrio que possa garantir melhor comportamento de aplicações de tempo real sem sacrificar o aproveitamento do sistema.

- Medições utilizando tarefas de rede e disco separadamente; Os resultados obtidos pela execução da aplicação TR9 ilustram o comportamento do sistema resultante da interferência das tarefas envolvendo dispositivos de rede e disco em conjunto. Não foi determinado qual destes dispositivos provocam maior interferência sobre aplicações de tempo real.

- Medições utilizando novas versões do kernel do Linux; Sabendo que a próxima versão do Linux traz modificações que objetivam o melhor comportamento de tarefas de tempo real brandos, experimentos sobre esta nova versão poderiam identificar o perfil de aplicação que obtém vantagens com o uso de um kernel com maior quantidade de pontos de preempção.

- A análise de versões do Linux para tempo real; Uma das questões de maior discussão envolvendo Linux e tempo real reside na eleição da versão deste sistema que provê melhor comportamento temporal às suas aplicações. Como cada uma delas adota uma estratégia diferente, voltada para pontos específicos a serem tratados em um sistema operacional de tempo real, experimentos no sentido de determinar o perfil das aplicações com melhor comportamento para cada versão poderiam fornecer informações de utilidade para desenvolvedores e projetistas.

Referências Bibliográficas

[1] AUDSLEY, N. C., TINDELL, K., BURNS, A., AND WELLINGS, A. J. Deadline Monotonic Scheduling Theory and Application. In *Control Engineering Practice* (February 1993), vol. 1, pp. 71 – 78.

[2] BAKER, T. P. Stack-Based Scheduling of Realtime Processes. vol. 3, pp. 67 – 90.

[3] BECK, M., BOHME, H., DZIODZKA, M., KUNITZ, U., MAGNUS, R., AND VERWORNER, D. *Linux Kernel Internals*, 2nd ed. Addisson-Wesley, 1999.

[4] B.NICHOLS, BUTTLAR, D., AND FARRELL, J. *Pthreads Programming*, 1st ed. O'Reilly and Associates, Inc., California, USA, 1996.

[5] BOVET, D. P., AND CESATI, M. *Understanding the LINUX KERNEL*. O'Reilly, 2001.

[6] BURNS, A., AND WELLINGS, A. *Real-Time Systems and Programming Languages*, 2st ed. Addison-Wesley, 1997.

[7] DANKWARDT, K. Fundamentals of Real-Time Software Design. December 2000.

[8] DE OLIVEIRA, R. S. *Escalonamento de Tarefas Imprecisas em Ambiente Distribuído*. Tese de doutorado, Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, 1997.

[9] DE OLIVEIRA, R. S. Sistemas Operacionais como programas concorrentes. In *ERAD 2002 2a Escola Regional de Alto Desempenho* (São Leopoldo, RS, Brasil, Janeiro 2002), pp. 139–177.

[10] FARINES, J. M., FRAGA, J. S., AND DE OLIVEIRA, R. S. *Sistemas de tempo real*, 1st ed. Escola de Computação 2000, 2000.

[11] GILBERT, D. Interrupt latency results for Linux <http://www.zip.com.au/ akpm/linux/intlat/intlat-disk.html>. Tech. rep., 2000.

[12] KOPETZ, H., AND VERISSIMO, P. *Real Time and Dependability Concepts*. 1993.

[13] L. SHA, R. RAJKUMAR, J. P. L. Priority Inheritance Protocols: An approach to Real-Time Synchronization. vol. 39, pp. 1175–1185.

[14] L. SHA, S. S. S. Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real_Time Systems. In *Proceedings of IEEE* (January 1994), vol. 82, pp. 68 – 82.

[15] LEHOCZKY, J., SHA, L., AND DING, Y. The Rate Monotonic Scheduling Algorithm: Exact Characterization And Avarage Case Behavior. *Proceedings of the IEEE Real-Time Systems Symposium 25*, 1 (Jan. 1989), 166–171.

[16] LEUNG, J. Y. T., AND WHITEHEAD, J. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. vol. 2, pp. 237–250.

[17] LIU, C. L., AND LAYLAND, J. W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. pp. 46–61.

[18] MAXWELL, S. *Kernel do Linux*, 1st ed. Makron Books, 2000.

[19] MONTEZ, C. B. *Um Modelo de Programação e uma Abordagem de Escalonamento Adaptativo usando RT-Corba*. Dissertação (doutorado), Universidade Federal de Santa Catarina, Santa Catarina, Brasil, 2000.

[20] PICCIONI, C. A., TATIBANA, C. Y., AND DE OLIVEIRA, R. S. Trabalhando com Tempo Real em Aplicações Sobre o Linux. Tech. rep., Universidade Federal de Santa Catarina, 2001.

[21] RAMAMRITHAN, K., AND STANKOVIC, J. A. Scheduling Algorithms and Operating Systems Support for Real- Time Systems. In *Proceedings of IEEE* (January 1994), pp. 55 – 67.

[22] RUBINI, A. *LINUX DEVICE DRIVERS*. O'Reilly, São Paulo, 1999.

[23] SHIN, A., AND RAMANATHAN, P. Real Time Computing: A New Discipline of Computer Science and Engineering. vol. 82, pp. 6 – 24.

[24] STANKOVIC, J. A. Misconeptions About Real-Time Architeture: A Serious Problem for Next Generation System. vol. 21, pp. 10 – 19.

[25] STANKOVIC, J. A. Misconeptions About Real-Time Architeture: A Serious Problem for Next Generation System. vol. 21, pp. 10 – 19.

[26] TANENBAUM, A. S. *Modern Operating Systemsl*, 2nd ed. Prentice Hall, 2000.